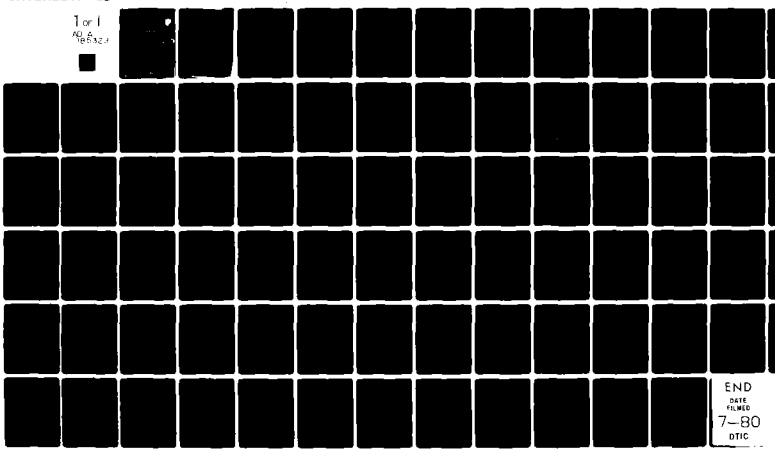
AD-A085 323 GENERAL ELECTRIC CO ARLINGTON VA F/G 9/2
EVALUATION OF SOFTWARE LIFE CYCLE DATA FROM THE PAVE PAWS PROJE--ETC(U)
MAR 80 B CURTIS, S B SHEPPARD, E KRUESI F30602-77-C-0194
UNCLASSIFIED RADC -TR-80-28 NL



END
DATE FILMED
7-80
DTIC

LEVEL II

RADC-TR-80-28
Final Technical Report
March 1980

EVALUATION OF SOFTWARE LIFE CYCLE DATA FROM THE PAVE PAWS PROJECT

General Electric Company

Bill Curtis
Sylvia B. Sheppard
Elizabeth Kruesi

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-80-28 has been reviewed and is approved for publication.

APPROVED: Donald F. Roberts

DONALD F. ROBERTS
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIS), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-80-28	2. GOVT ACCESSION NO.: AD-A085323
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): EVALUATION OF SOFTWARE LIFE CYCLE DATA FROM THE PAVE PAWS PROJECT	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. 1 Sep 77 — 30 Nov 79
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Bill Curtis, Sylvia B. Sheppard, Elizabeth Kruesi	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0194
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Electric, Suite 200, 1755 Jefferson Davis Hwy, Arlington VA 22202	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: J.O. 25280103
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: March 1980
	13. NUMBER OF PAGES: 72
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Don Roberts (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Modern programming practices; Top-down design; Structured coding; Program support library; Program design language; HIPO charts; Chief programmer teams; Structured walkthroughs; Software quality assurance; Software error data; Software management; Software database; Structured programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Data were collected over the development cycle of the PAVE PAWS software development project. This project was designed to be a technology demonstration of modern programming practices. The practices studied on this project included:

(1) Top-down design and implementation,
(2) Structured coding and precompilers,
(3) Program support library

(Cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(4) Program design language and HIPO charts,
(5) Chief programmer teams,
(6) Structured walkthroughs,
(7) Independent test and quality assurance groups.

The data available for evaluating this project included personhours, trouble reports, compiler summaries, code progression and durability charts from the library, management summaries, and personnel profiles. Only personhours and trouble reports were collected throughout the project. The PAVE PAWS project resulted in 211,000 lines of code which took 1,133 personmonths of effort to produce. The total effort, productivity, and number of errors associated with the project were found to be typical of similar sized software development efforts when plotted into the RADC database. After some redefinition of error categories, the relative frequency of the types of errors experienced on PAVE PAWS was found to be similar to other projects in the RADC error database. It was concluded from these data and our previous study on the LSDB project that modern programming practices are not miraculous productivity aids. Rather, these practices represent sound management principles which make software development more manageable and the prediction of project outcomes more accurate. Recommendations are also made for methods of collecting software life cycle data in future studies.

Table of Contents

List of Tables

List of Figures

EVALUATION

This effort was initiated to analyze data collected during two software development projects, in order to assess the impact of modern programming practices on software productivity, quality and cost. The effort was undertaken in response to requirements defined in TPO-5, Software Cost Reduction and subthrust Software Data Collection and Analysis. A goal of that subthrust is to establish baselines for software data related to software costs, errors, productivity, quality and maintenance. The baselines will be useful in identifying factors and characteristics that contribute to the above factors in either a negative or positive sense. The results of the data analysis performed on the first effort are documented in an interim report, RADC-TR-80-6, Volumes I and II, "A Matched Project Evaluation of Modern Programming Practices" dated February 1980.

This report documents the results of analysis of data collected during the development of PAVE PAWS software. A number of modern programming practices were used on the PAVE PAWS software development. Data pertaining to program compilations, errors and productivity was analyzed and compared to similar data collected from other development efforts. The results, in conjunction with other data previously obtained will further the development of reliable baselines for gauging future software development efforts.

The report also presents a model for software data collection and analysis that will significantly aid future research in this area.

DONALD F. ROBERTS
Project Engineer

1. INTRODUCTION

1.1 Purpose for This Research

In 1973 Boehm chronicled the tremendous impact of software on the cost and reliability of advanced information processing systems. Recently, DeRoze and Nyman (1978) estimated the yearly cost for software within the Department of Defense to be as large as three billion dollars. DeRoze (1977) reported that perhaps 115 major defense systems depend on software for successful operation. Nevertheless, the production and maintenance of software for Defense is frequently inefficient.

In an effort to improve both software quality and the efficiency of software development and maintenance, a number of techniques have been developed as alternatives to conventional programming practices. These modern programming practices include such techniques as structured coding, structured design, program support libraries, and chief programmer teams (W. Myers, 1978; Tausworthe, 1979). The New York Times project implemented by IBM (Baker, 1972) was lauded as a successful initial demonstration of these techniques. Yet, some problems appear to have resulted from an early release of the system (Yourdon Report, 1976). Considerable variability has been reported in subsequent studies sponsored by Rome Air Development Center (RADC) on the effect of these techniques for various project outcomes (Belford, Donahoo, & Heard, 1977; Black, 1977; Brown, 1977; Curtis & Milliman, 1979). Many evaluations have rested solely on subjective opinions obtained in questionnaires. There is a critical need for empirical research evaluating the effects of modern programming practices on software development projects.

Recently, Curtis and Milliman (1979) evaluated the effects of modern programming practices using objective project data collected by RADC. They studied the effectiveness of the ASTROS plan (Lyons & Hall, 1976) developed jointly by RADC and the Space and Missile Test Center (SAMTEC) at Vandenburg Air Force Base. This system provides guidelines for applying structured design and testing, HIPO charts, chief programmer teams, structured coding, structured walkthroughs, and a program support library to software development projects. In order to test the utility of these techniques, two non-real-time

development projects sponsored by SAMTEC from a system which provides control and data analysis for missile launches were chosen for a quasi-experimental comparison. The Launch Support Data Base (LSDB) was developed under the guidelines of the ASTROS plan, while the Data Analysis Processor (DAP) was developed using conventional techniques.

Results indicated that the performance of the LSDB project was comparable to that of similar-sized software development projects on numerous criteria. The amount of code produced per man-month was typical of conventional development efforts. Nevertheless, the performance of the LSDB project was superior to that of the DAP project. That is, the LSDB project team produced higher quality code with less programming effort and experienced fewer post-development errors than the DAP project. Thus, Curtis and Milliman concluded that the benefits of modern programming practices were often limited by the constraints of environmental factors such as computer access and turnaround time. They believed that further evaluative research would be required before confident testimonials could be given to the benefits of modern programming techniques. Nevertheless, the results of their study suggest that future evaluations will yield positive results if constraints in the development environment are properly controlled.

In an attempt to gain further evaluative data on the effectiveness of modern programming practices, RADC instituted a data collection system on the software development portion of the PAVE PAWS system development effort. The software development effort associated with the PAVE PAWS project was substantially larger than that studied in the SAMTEC-ASTROS project and offered an opportunity to study modern programming practices in a multi-team environment. This report presents the results of analyzing the PAVE PAWS data.

1.2 PAVE PAWS System Description

Descriptions in this and the following section were largely based on the final report of the data collection system filed by Raytheon (1979). PAVE PAWS is a fixed base Phased Array Warning System utilized for the detection and tracking of Submarine Launched Ballistic Missiles (SLBM's) which penetrate the radar coverage. It consists of two phased array warning sensors located at Otis AFB, MA and Beale AFB, CA. The primary mission of PAVE PAWS is to provide the NORAD Cheyenne Mountain Complex (NCMC) with credible warning of SLBM attacks, including estimation of launch and impact points and times. As a secondary mission

PAVE PAWS supports the USAF SPACETRACK System with earth satellite vehicle surveillance, tracking, and data collection as requested by NCMC.

The system includes six display consoles which are used for systems operations, monitoring and control, missile warning operations, SPACETRACK operations, training, and maintenance control. Over thirty different display formats are independently selectable at the display consoles in order to provide complete flexibility in monitoring and controlling the system. Because PAVE PAWS is an on-line system which is intended to be operational at all times, the data processing system contains redundant hardware throughout. In the event of a hardware or software fault, hardware is automatically reconfigured to eliminate the fault and resume the primary mission within 8 seconds. The data processor (duplex CDC CYBER 174's) communicates with one of two MODCOMP mini-computers which interface directly with the radar hardware.

In addition to the software to perform the primary and secondary missions of PAVE PAWS, the system includes a simulation facility capable of operating concurrently with the operational software which provides the full range of mission, threat, communications, and radar stimuli to that software. Object trajectories, radar cross sections, launch and impact points, communications messages, radar environmental effects, and event timing can be simulated under user specification. The system also records real-time data pertinent to the performance of the primary and secondary missions and provides data reduction capabilities for a wide variety of recording formats.

1.3 Software Development Technology

The PAVE PAWS system specification required that all software be developed in a modular manner utilizing top-down structured programming with clear interface specifications to provide management visibility. Where practical all software was to be coded in Jovial. The use of the JOVIAL statements DIRECT/JOVIAL was not permitted. Exceptions in the use of Jovial were allowed for highly used algorithms, I/O interface routines, and the operating system/operating system interface routines which could be coded in a low level language such as micro code, machine, or assembler for more efficient usage of the data processing hardware. Fortran was allowed for use in the Radar Controller.

A number of modern programming practices were used in the PAVE PAWS software development effort. These practices

included:

- Use of a comprehensive Program Support Library sys
- Top-down design and implementation
- Structured coding, including the use of language precompilers
- Use of Program Design Language and HIPO charts
- Chief Programmer Team operations
- Use of structured walkthroughs for reviewing desig and code
- Development of independent test and quality assura groups

These techniques will be described in greater detail below.

1.3.1 Program Support Library (PSL). The PAVE PAWS PS is a programming tool specifically designed to support and enforce top-down, structured programming techniques. This requires a program storage and maintenance capability which allows considerable program segmentation and a precompiler which allows the commercial Jovial, Compass, and Iftran languages to include the necessary structured constructs. T PSL also accommodates a structured Program Design Language (PDL).

The PAVE PAWS PSL was designed to support an orderly progression of software from a development environment through integration and test into a delivered product throu use of a multi-level hierarchical configuration control. Software segments are entered into the library using a user specified name (up to 40 characters long) at a user specifi level. Since each level of the library is distinct from other levels, the same software element may appear in the library at several different levels. Thus, to completely identify an item in the library it is necessary to specify both the name and level. This provides a simple mechanism for parallelism in development, error correction, and versi modification. Within the PSL, seven hierarchical library levels were defined. Starting with the highest level in th PSL these levels include:

- DEL - software which is in the field
- FRZ - software which has been qualified

- TST - software undergoing qualification test
- FIX - software corrections for TST level
- INT - software undergoing integration test
- CPT - software undergoing group test
- PRG - software under development/unit test.

A program element is ready to migrate to the next highest PSL control level when it has satisfied a predefined qualification criteria and is to be placed under more stringent change control. This migration is effected within the PSL by a raise level command (XMIT) which moves the element to the specified level. In order to facilitate changes to segments once they have been raised to higher levels, the PSL includes a feature called "automatic drawdown". This feature allows library operations to be addressed to a specific library level and "drawdown" a copy of a program element which is under configuration control at a higher level. Changes can be made to this program element at the lower level, but it can only migrate back up to its original level by satisfying the qualifications of each successively higher level of the PSL. A detailed discussion of the authorization system required to accomplish change control is presented in Raytheon (1979).

Code progression and durability charts could be requested of the PSL and used as management information tools. The code progression chart is organized as a CPCG/level matrix which indicates how much effective code exists (using drawdown as necessary) at each level of the library. Thus code which exists at the INT level of the library, also "effectively" exists at the PRG and CPT levels as well. Since each of the library levels represents a different testing benchmark, this report allows management to answer questions like "How much code has been written?", "How much code has reached functional test?", and "How much code has been integrated?".

The code durability report acknowledges the fact that segments which have already been changed at lower library levels represent a discount to the amount of code under configuration control at higher PSL levels in the code progression report. The accounting mechanism employed in the durability report ignores segments which have been "drawndown" for further change at a lower level. The durability report shows management that it is dangerous to consider a segment as having been successfully integrated when at the INT level of the library if it is simultaneously

undergoing change at the PRG level. To calculate "durable" lines of code, the PSL counts each unique segment only once. This count is made at the lowest level of the library where the segment appears. The value of this report lies in complementing the progression report in allowing management to answer questions such as "How stable (durable) is the code that has been developed?" and "How much effort remains?".

1.3.2 Top-down programming/segmentation. Top-down programming is based upon a technique of designing and implementing software by specifying the top level functions first (G. Myers, 1978; Stevens, Myers, & Constantine, 1974; Yourdon & Constantine, 1979). The details of each of those functions and the specification of additional subfunctions are then developed through successive iterations until the entire problem is fully developed. Throughout this process the amount of design or code allowed in a single component is purposely kept small to make it more manageable. This is accomplished by treating total functions or subfunctions as "black box" modules with known input and output requirements. This modularization (Parnas, 1972) is reflected in the PSL through program segmentation. A segment of program code can identify a needed function by using an INCLUDE statement. This named function can then be dealt with independently, and it may itself utilize INCLUDE statements to identify and define even lower level functions. In this way a program is developed as a set of single page segments which fit together in a program structure or hierarchy.

The Top-Down aspect of software development is enforced by identifying each segment placed in the library as either a top-segment (i.e., the top-level of an independently compiled program) or as an INCLUDEed segment (one which is simply a lower-level part of some program). As top-level segments are entered into the library and INCLUDE statements are encountered, stubs are generated to act as position holders until actual code is provided. A program stub identifies the need for code to perform the named function, it reserves the name for that function, and since it is part of some already existing program it specifies the implementation language for that function. The top-down ordering of software development is enforced by requiring that INCLUDEed segments cannot be added into the PSL library unless they are replacing a stub. In addition, since stubs represent unimplemented software segments, the number of stubs in a program can be used as a measure of status or progress.

The development of large software systems presents a substantial challenge in the management of system components. The allocation of system requirements to individual Computer Program Configuration Items (CPCIs) is an important function

because from that point forward each CPCI will be managed with a certain degree of autonomy. Managing these components includes estimating and planning the effort involved, allocating resources, assessing and reporting status, financial management and reporting, and the resolution of technical problems.

Three principles were observed in defining CPCIs on PAVE PAWS in order to establish an effective subdivision of the total software effort:

1) CPCI responsibility should not cross the corporate boundaries of the prime and subcontractors

2) CPCIs should not cross computer boundaries within the system hardware

3) Software systems which are executed separately should be separate CPCIs

Below the CPCI level, software is next broken into Computer Program Configuration Groups (CPCGs) and Computer Program Components (CPCs). CPCGs are generally structured along major functional lines within a CPCI while CPCs represent individual programs. This structuring of the software is important because it forms the basis for allocating system requirements to software, identifying interface control definitions, subdividing design and development responsibilities, and making personnel assigments. In short, a well understood software structure allows a software project to be effectively managed.

1.3.3 Structured Coding. Structured coding requires the use of a standard set of program control statements and at the same time precludes the use of explicit branching statements. In order to provide the standard set of control statements for Jovial, Compass, Iftran, and PDL, the PSL includes a pre-compiler which accepts the structured source statements and converts them into traditional control forms which are processed by the appropriate compiler. Figure 1 shows both the logical and coded form of each of the PAVE PAWS standard control constructs. The requirement to provide a separate statement to end each of the constructs provides a closure mechanism for the generation of indented listings.

1.3.4 Structured documentation aids. Hierarchical Input-Process-Output (HIPO) charts are diagramatic representations of the operations performed on the data by each major unit of code (Katzen, 1976; Stay, 1974). A HIPO chart is essentially a block diagram showing the inputs into a functional unit, the processes performed on that data within the unit, and the

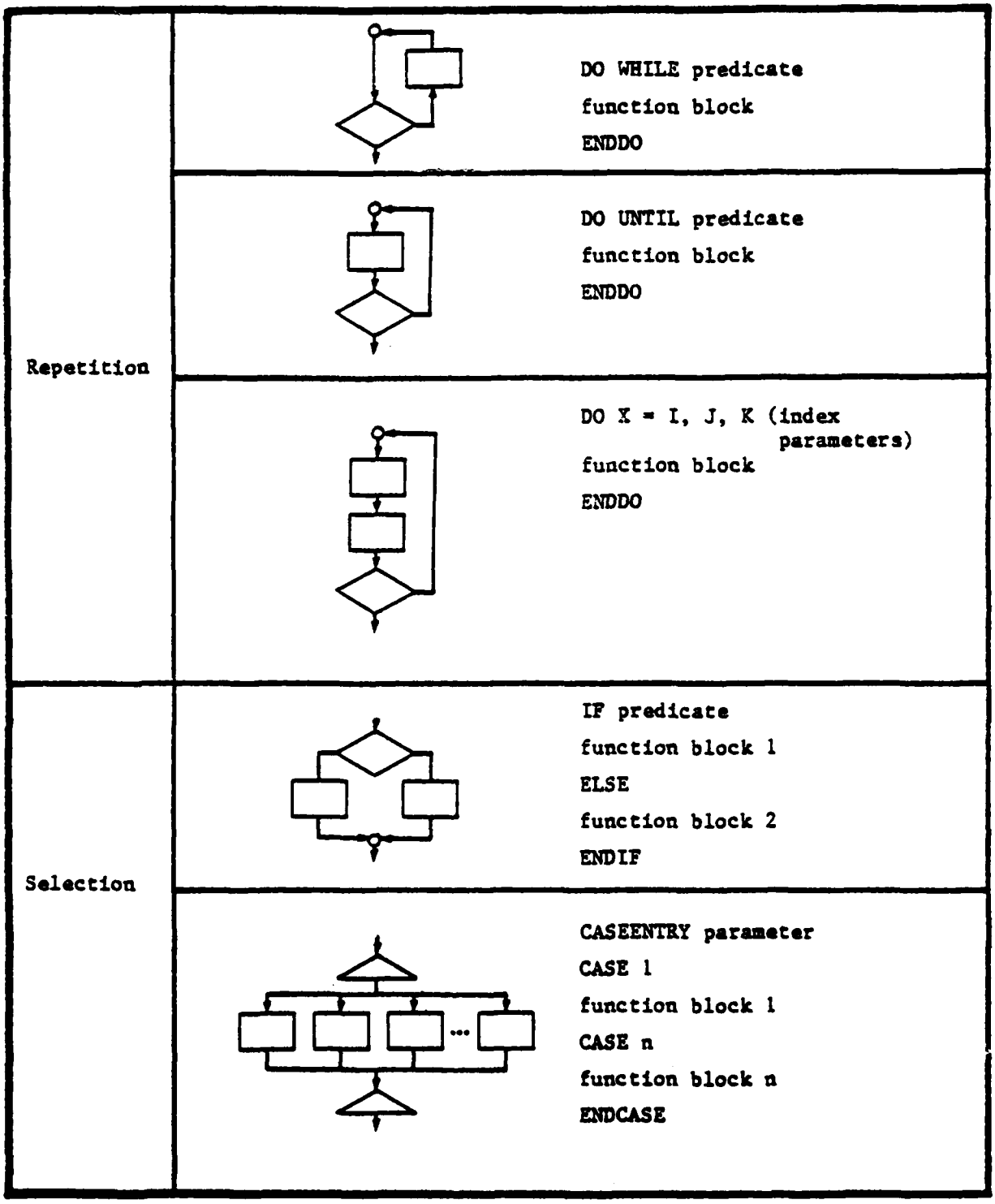


Figure 1. Logical and coded forms of structured control flow

output from the unit. Typically there is one HIPO per functional unit, with the processing in the unit being expanded to new HIPOs until the lowest level of detail is reached. The hierarchical relationships among the HIPO charts are displayed in a Visual Table of Contents. Examples of these documentation aids are presented in Appendix B.

1.3.5 Chief programmer teams. Chief programmer teams are organized so that functional responsibilities such as data definition, program design, and clerical operations are assigned to different members (Baker, 1972, Baker & Mills, 1973; Barry & Naughton, 1975). This approach results in better integration of the team's work, avoiding the isolation of individual programmers that has often characterized programming projects. The chief programmer team is made up of three core members and optional support members who are programmers. The three core members are:

- Chief programmer - is responsible to the project manager for developing the system and managing the programming team. He or she carries technical responsibility for the project including production of the critical core of the programming system in detailed code, direct specification of all other codes required for system implementation, and review of the code integration.

- Backup programmer - supports the chief programmer at a detailed task level so that he or she can assume the chief programmer's role temporarily or permanently if required.

- Librarian - assembles, compiles, and link-edits the programs submitted by project programmers. The librarian is responsible for maintaining any records not maintained by the program support library.

1.3.6 Structured walkthroughs - A structured walk-through is a review of a developer's work (program design, code, documentation, etc.) by fellow project members invited by the developer. Not only can these reviews locate errors earlier in the development cycle, but reviewers are exposed to other design and coding strategies. A typical walk-through is scheduled for one or two hours. If the objectives have not been met by the end of the session, another walkthrough is scheduled.

While a broad range of people often including customers were invited to attend design walkthroughs, only a few other programmers were invited to attend code walkthroughs. During a walkthrough reviewers are requested to comment on the

completeness, accuracy, and general quality of the work presented. Major concerns are expressed and identified as areas for potential followup. The developer then gives a brief tutorial overview of his work. He next walks the reviewers through his work step-by-step, simulating the function under investigation. He attempts to take the reviewers through the material in enough detail to satisfy the major concerns expressed earlier in the meeting, although new concerns may arise.

It is the responsibility of the developer to ensure that the points of concern raised are successfully resolved and reviewers are notified of the actions taken. It is important that walkthrough criticism focus on error detection rather than fault finding in order to promote a readiness to allow public analysis of a programmer's work.

1.3.7 Independent test and quality assurance functions. A test organization separate from the software development group was created which was responsible for developing all test documentation and for conducting the tests. Further there was an organizational separation from the software development group of the group responsible for developing the Quality Assurance Program, including the establishment of project-wide procedures, implementation of a Trouble Reporting system, and providing regular assessments of status and forecasts for management consideration and action.

1.4 Data Collection Techniques

The intent of the Data Collection effort was to provide data which characterized the nature and environment of the software development activity together with information about the reasons underlying software changes. The collection of this data was accomplished in three ways:

1) Manual collection of project and personnel characteristics.

2) Automatic collection of software change data by the PAVE PAWS PSL.

3) Automatic recording and summarization of software change activity as part of a project-wide Trouble Report/Change Request (TR/CR) system.

Because the bulk of the software design and development had been completed by the time of contract award, the automatic collection of data was augmented by a one-time manual reconstruction of the existing TR/CR database.

Data were collected from IBM's software development effort on CPCI2 (Tactical Software), CPCI3 (Simulation Software), CPCI4 (Program Support Library), and CPCI5 (Data Reduction). No attempt has been made to discuss data from CPCI1 (PAVE PAWS Operating System), CPCI6 (Radar Control Software), or CPCI7 (Signal Processor Software), since they were developed by a contractor other than IBM and personhours and lines of code were not available for these CPCIs.

Six classes of data collected on this project are available for analysis in evaluating the use of modern programming practices. These types of data are:

- Personhours
- Trouble Reports
- Compile summaries
- Code progressions
- Personnel profiles
- Project summary forms

Unfortunately, only personhours and trouble reports seem to have been collected throughout the development cycle of the PAVE PAWS project. Figure 2 presents the time periods covered by each of the six classes of data. Most development work seems to have been completed by the Final Qualification Test held in June 1978. The nature of the data available for each class will be described separately.

1.4.1 Manual data collection. The following types of data were provided through the completion of forms by project personnel (the first three summaries appear in Appendix A). The General Contract/Project Summary provides general information about the size of the project (cost, people, software, and documentation) together with a high level technical description of the project. The Management Methodology Summary identifies management procedures utilized, the schedule for PDR's and CDR's, and an enumeration of the Air Force and Military Standards which apply. The Design and Processor Summary identifies the data processor configuration, the programming languages used, the standards followed, and the software technology utilized. Finally, Chief Programmer Team Profiles characterize the educational and work experiences of each of the teams on the PAVE PAWS software development project. Personhours for the system and for CPCI's 2, 3, 4, and 5 were collected from May

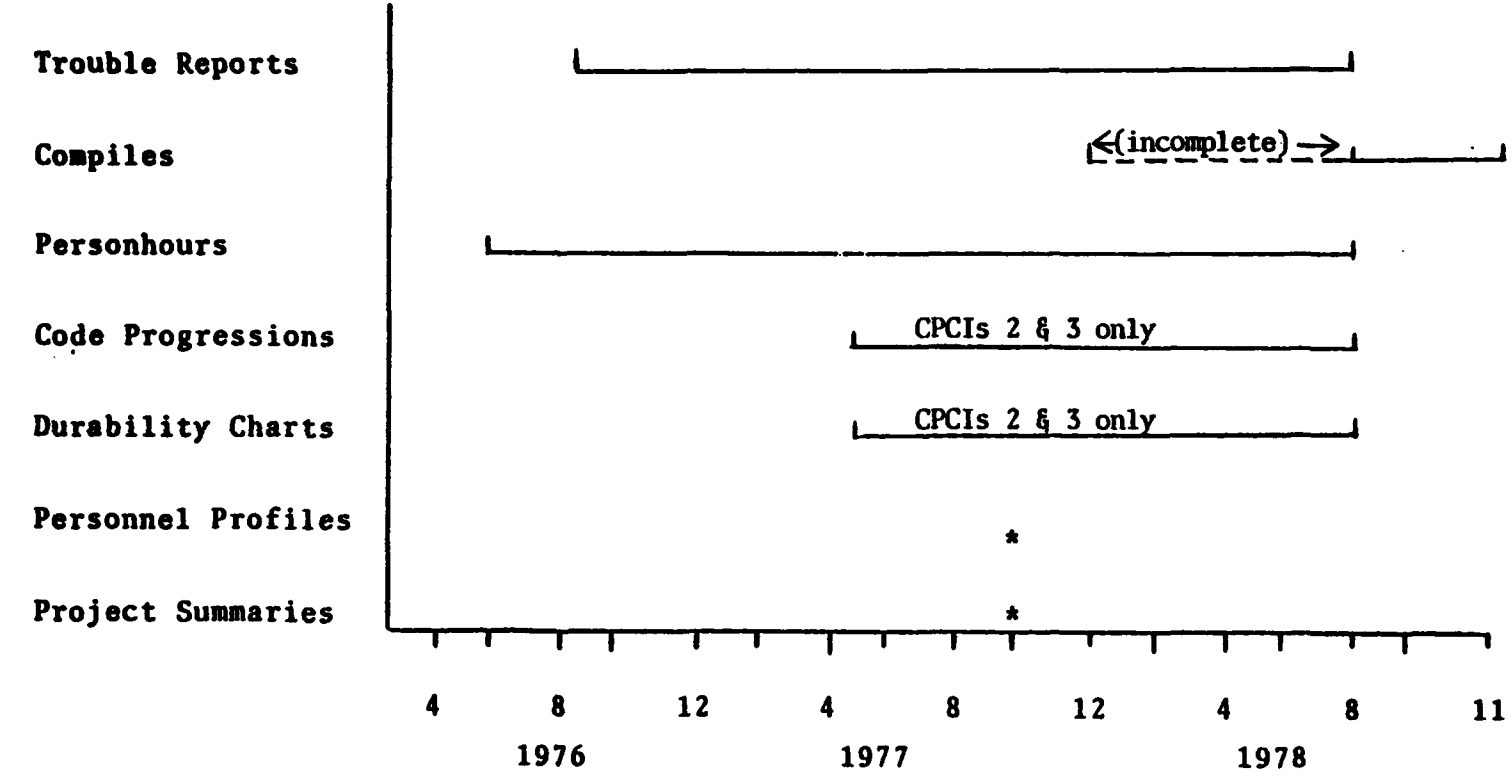


Figure 2. Data available from the PAVE PAWS project

1976 through June 1978. Personhours were broken into work categories for each CPCI.

1.4.2 Trouble reporting system. The Trouble Reporting System provided a report for each problem encountered in code which had reached the INT level of the PSL from August 1976 to June 1978. The Trouble Reports (TRs) were completed on standard form (Figure 3). These forms were collected manually and automated at a later date. The information includes a description of the problem, an error category, the CPCI, the CPCG, the priority of the change (emergency, urgent, or routine), and the level of the hierarchical library at which the change was made. A brief description of the error categories is presented in Table 1. The categories of interest are 1 to 14. Approximately 20% of the TR's have an "unknown" error code. Although there is a short description of the problem causing the report to be written, it would be difficult to accurately assign codes for the "unknown" group without knowing the software system.

1.4.3 Automated data collection. The Program Support Library programs were modified to read the compiler list output and determine compiler detected errors. A special data file was added to the PSL for the purpose of saving compiler detected errors. The contents of this data file were used as inputs to a report program on a weekly basis to produce the PSL error reports which were provided to RADC as part of the data collection effort. Impact on the PSL users was minimal, with one additional field required for compilation (compile reason code). These compile reason codes are described in Table 2.

The PSL produced compile listings for the period from January 1978 to November 1978. These included the program name, date, program edition, level, lines of code, and reason for the compile. The majority of the compiles (about 98%) occurred from June 1978 to November 1978. This period of time occurred after the major development effort was completed. Thus, these compiles are not representative of the major portion of the project. For example, the purpose of many of these compiles was to get printed listings of the code.

Charts of the code progressions for CPCI's 2 and 3 were produced for the period from April 1977 through June 1978. These charts present the amount of code that has been approved at each of four levels within the PSL over time. The data were used primarily in the management reporting system.

Durability charts are similar to code progressions

PAVE PAWS PROGRAM TROUBLE REPORT/CHANGE REQUEST

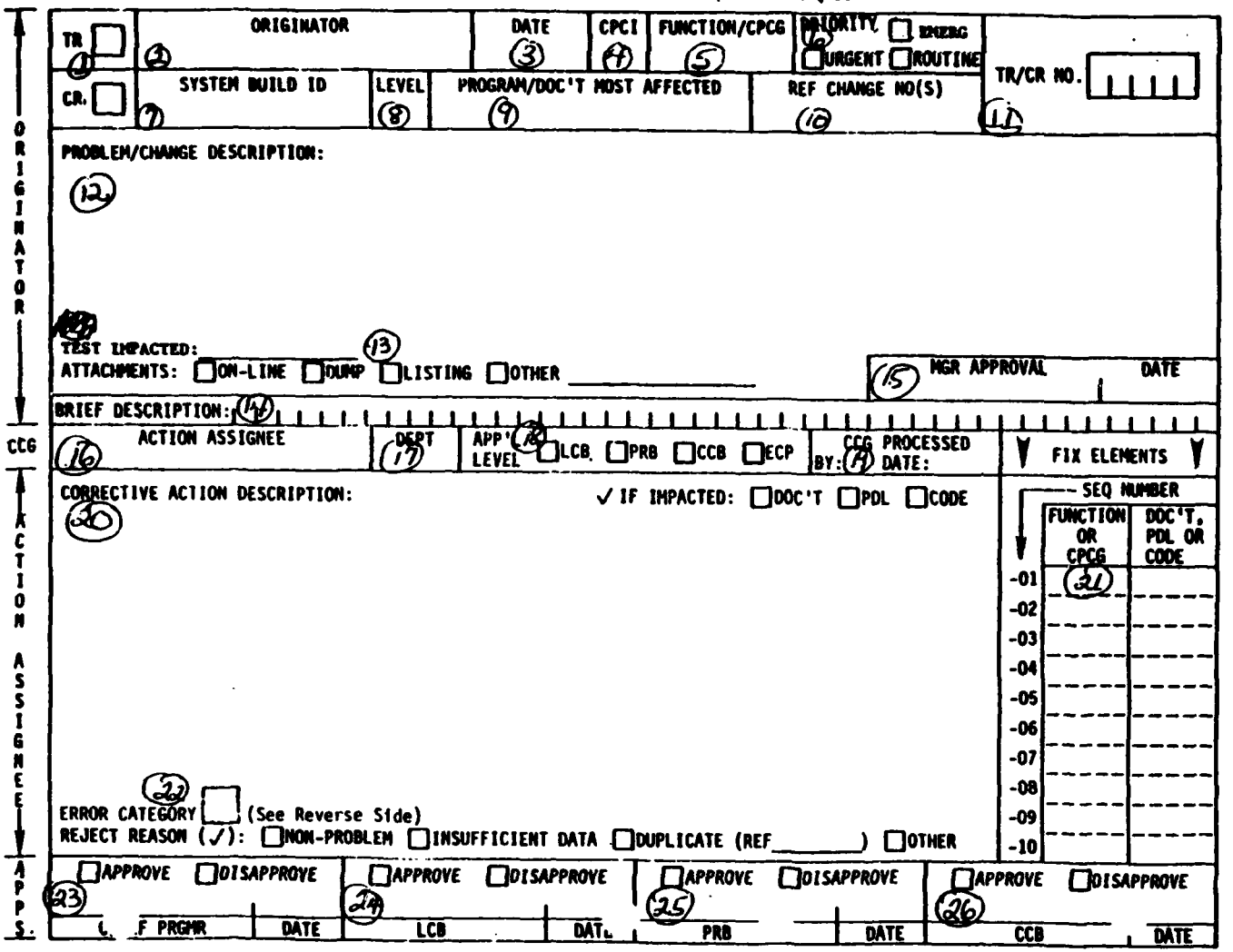


Figure 3. Trouble Report form

Table 1

PAVE PAWS Error Categories

Code	Title	Description
0	Unknown	
1	Computation	Error in implementation of equations
2	Logic	Error in decision logic
3	Data Base	Error in data base definition
4	Input/Output processing	Error in processing data items
5	Specified function not implemented	Missing code
6	Specified interface not implemented correctly	This could apply to hardware, operating system, other programs, common data area, etc.
7	Unspecified function required	Additional problem definition needed
8	Unspecified interface not satisfied	This could apply to hardware, operating system, other programs, common data areas, etc.
9	Memory/throughput optimization	Additional optimization required
10	Design modification/ enhancement	Change to current design
11	Documentation change only	Type C spec change/user manual/PDL
12	Keypunch	Mistake in keypunching
13	Deck setup	JCL Procedure error
14	Configuration	Build uses mismatched code, wrong IGS package in Build, etc.
15	Open	Not yet closed or categorized
16	Reject	Duplicate of another TR
17	Reject	Insufficient data to support TR
18	Reject	Non-problem
19	Reject	Other reason

Table 2

Compile Reason Codes

Label	Title	Description
INITIAL	Initial Program Compile	Used until the program compiles without compiler detected error
KEY	Keypunch Error	Used when keypunching errors are being corrected
SETUP	Deck Setup Error	Used when the compile is to correct a deck setup error such as using the wrong COMPOOL
COMP	Computational Error	Used when correcting computational errors such as the wrong sign or wrong trigonometric function
LOGICAL	Logic Error	Used when correcting logic errors such as NE instead of EQ
DATA	Data Base Error	Used when correcting data base errors such as tables not correctly initialized
IO	I/O Error	Used to correct errors in using the I/O facilities such as changing reads to puts or adding necessary WAIT statements
SFNI	Specified Function Not Implemented	Used to insert functions whose implementation has been deliberately delayed
SINI	Specified Interface Not Implemented	Used to insert interface code which has been deliberately deferred
FUNCHG	Unspecified Function	Used to implement new or changed functions
INTCHG	Unspecified Interface	Used to implement new or changed interfaces
MEMOPT	Memory Optimization	Used to compile changes made to improve core memory utilization
CPUOPT	CPU Time Optimization	Used to compile changes made to improve CPU utilization
LOGOPT	Logic Simplification	Used to compile changes made to the program to make the logic easier to understand
COMMENT	Comment	Used when the compile is to verify the legality of comments

Table 2. (Cont'd)

Abbreviation	Title	Description
LIST	Extra Listing Required	Used when the compile is to obtain an extra listing or an additional listing feature e.g., generated code
VERIFY	Object Module Verification	Used when the purpose of the compile is to guarantee that the object and source code match. This code should also be used when a common include has been changed in another program
COMPILER	Compiler Error	Used when investigating or correcting internal computer errors
PPOS	Operating System Error	Used when correcting operating system errors
PSL	PSL Internal Errors	Used when correcting PSL internal errors

except that they present the highest level of the PSL at which a program is resident without having been drawn down for additional changes at a lower level. Thus, if a program has been approved at the TST level, but it has been drawn down to the PRG level for further changes, the durability chart will represent this program at the PRG level. Similar to code progressions these charts present data collected between April 1977 and June 1978.

1.4.4 Other sources of data. Unfortunately the PAVE PAWS source code was unavailable due to security reasons. However, three additional sources of information regarding the development of this code are available. These sources include the Green Sheets which describe the standards observed on the project. They provided a means of communication among project personnel on coding practices. The Configuration Management Plan described the use of the PSL in controlling the development of the code. Finally the IBM Software Quality Assurance Plan described the methods used to manage code quality.

RADC has compiled several databases of information relevant to software development against which the performance of the PAVE PAWS project can be assessed. RADC has collected development data over a large number of systems, including military and commercial software projects (Duvall, 1978; Nelson, 1978). These data were collected in an attempt to establish baselines and parameters typical of the software development process. Some of the variables against which the PAVE PAWS data can be compared are listed in Table 3. RADC has also sponsored a number of studies which have provided detailed categorizations of error types (Baker, 1977; Curtis & Milliman, 1979; Fries, 1977; Rye et al., 1977; Thayer et al., 1976; Williams et al., 1977) against which the PAVE PAWS error categories can be compared

Table 3

Variables in the RADC Database

Variable	Definition
Program Size	The total number of lines of source code in the delivered product. This count includes declarations, internal program data, and comment lines. It does not include throwaway or external data.
Project Effort	The number of man-months required to produce the software product, including management, design, test, and documentation.
Project duration	The number of months elapsed during the development phase minus dead time such as work stoppages.
Errors	The number of formally recorded software problem reports for which a correction was made during the period covered by the project. This does not include errors from the development portion of the project, but rather from testing through integration.
Derived parameters	Ratios obtained from other variables: a. Productivity = Size/Effort b. Average Number of Personnel = Effort/Duration c. Error Rate = Errors/Size

2. RESULTS

The analysis of the data from the PAVE PAWS project will be presented in two sections. The first section will relate to productivity, while the second section will present analyses of the PAVE PAWS error data. Analyses in the first section will include:

- descriptive data on lines of code and personhours
- productivity comparisons between PAVE PAWS and other projects
- descriptive data from the PSL database

Analyses in the section on error data will include:

- descriptive data on the PAVE PAWS error categories
- comparisons to error data from other projects

A major goal of this project was to use the data available to determine the effectiveness of various modern programming practices separately and in combination. Unfortunately such analyses are not possible from the PAVE PAWS data. In order to determine the effectivness of separate practices, data are required which compare project outcomes that are attributable to 1) the use versus nonuse (or degree of use) of a particular practice, 2) the use of a practice singularly or in combination with sets of other practices, and 3) environmental limitations on the practices employed. The first two types of data were not available since all project members were expected to observe all programming practices and standards throughout development. Differences might be detected if there were an indication of which team developed which sections of code and how these teams may have differed in their adherence to practices. However, no information was available which allowed this determination. The third type of data is available only indirectly by comparing the PAVE PAWS data to the RADC software database. Yet, without additional data on the factors which affected performance in these other projects this type of analysis is only approximate. Thus, assessments of the effectiveness of individual programming practices must rest on subjective reports provided by the development personnel. Comparisons to other projects must also be viewed warily since Curtis and Milliman (1979) demonstrated that the benefits of programming practices must be interpreted within

the constraints placed on their effectiveness by factors in the development environment. Nevertheless, the data presented in this report will be of heuristic interest even though they are insufficient for determining the measurable benefits of modern programming practices.

2.1 Productivity Analyses

2.1.1 Lines of code. As is evident in Table 4, the four CPCIs developed by IBM on the PAVE PAWS project consisted of approximately 211,000 lines of code. Thus, the final system is substantially larger than the original estimated 135,000 card images reported by project personnel in the General Contract/Project Summary (Appendix A). It is possible, however, that the number of instructions originally estimated did not include such lines as comments which would be included in counting the total lines of code.

It is evident from Table 4 that CPCI2 is composed of eight CPCGs some of which are as large as CPCIs 3, 4, and 5. Unfortunately, lines of code for the CPCGs in CPCIs 3, 4, and 5 were not available.

2.1.2 Personhours. The personhours expended in developing CPCIs 2 through 5 are presented in Table 5. The 169,888 hours for the total project represents 1133 person-months of effort, using a standard of 150 hours per person-month. This figure is quite close to the 1089 personmonths estimated for the project (General Contract/Project Summary, Appendix A). In preparing this table it was assumed that the 24,415 hours attributed to system level development of the four CPCIs was involved in preparing some support software (BLD, COMP, JOV, PROC, STP, and SYST) which was not defined as part of the four CPCIs. These hours were listed under code and integrate, but this assumption may be in error. The relative hours devoted to the development of each CPCI were consistent with the size of the code comprising the CPCI. Figure 4 presents the chronological personpower loadings by month and CPCI throughout the PAVE PAWS project.

2.1.3 Productivity comparisons with other projects. Nelson (1978) has produced a number of regression plots of delivered source lines of code against various project outcomes for projects in the RADC database. These scatter-plots allow a comparison of outcomes among projects while controlling for project size. Figure 5 presents the scatter-plot for total personmonths of effort versus delivered source lines of code. Datapoints for the total PAVE PAWS project and each CPCI have been separately plotted into the figure. It is evident that the number of personmonths required to

Table 4

Lines of Code by CPCI and CPCG

CPCI	Title	Lines of Code	
		CPCG	CPCI
2	Tactical Software		139,000
	RTM - Real Time Monitor	13,200	
	MCTL - Mission Control	5,200	
	SCM - Satellite Catalogue Manager	10,300	
	RAM - Radar Manager	12,900	
	TRCK - Track	16,100	
	DISP - Displays	45,900	
	COMM - Communications	8,700	
	TGDB - TIMEX Global Database	26,700	
3	Simulation Software		29,000
	DPCS - Data Processing Database		
	RTSM - Real Time Simulation		
	SGDB - SIMEX Global Database		
	TSG - Target Scenario Generation		
4	Support Software		16,000
	PSL - Program Support Library		
	LPC - Precompiler		
	MREP - PSL Management Reports		
5	Data Reduction		27,000
	DTRD - Data Reduction		
	PRNT - Print		
	STRP - Strip		
	SORT - Sort		
	LRID - Logical Record ID		
Total			211,000

Table 5

Personhours by Work Category and CPCI

Work Category	System	CPCI 2	CPCI 3	CPCI 4	CPCI 5	Total
System Engineering	9507	3393	1040	0	0	13940
Production Specification	4657	1555	0	0	0	6212
Detail Design	0	9662	2889*	2097*	3609*	18257
Code and Integrate	24415	55004	9052	5329	7872	101672
User Manual	0	0	289	0	0	289
Testing	3816	19836	2326	1594	1946	29518
Total Personhours	42395	89450	15596	9020	13427	169888
Total Personmonths	283	596	104	60	90	1133

*Includes personhours devoted to production specification

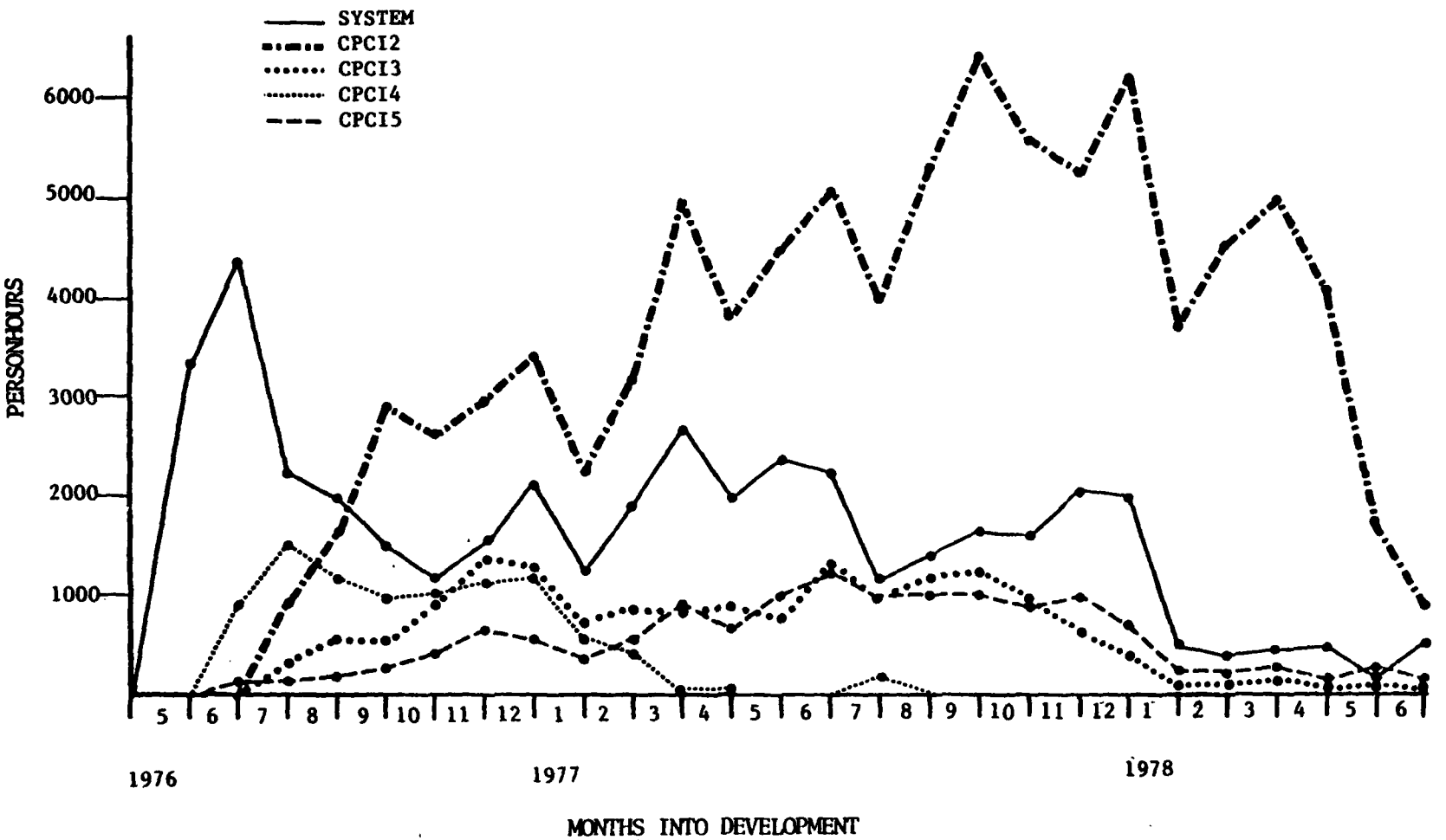


Figure 4. Chronological personhour loadings by CPCI

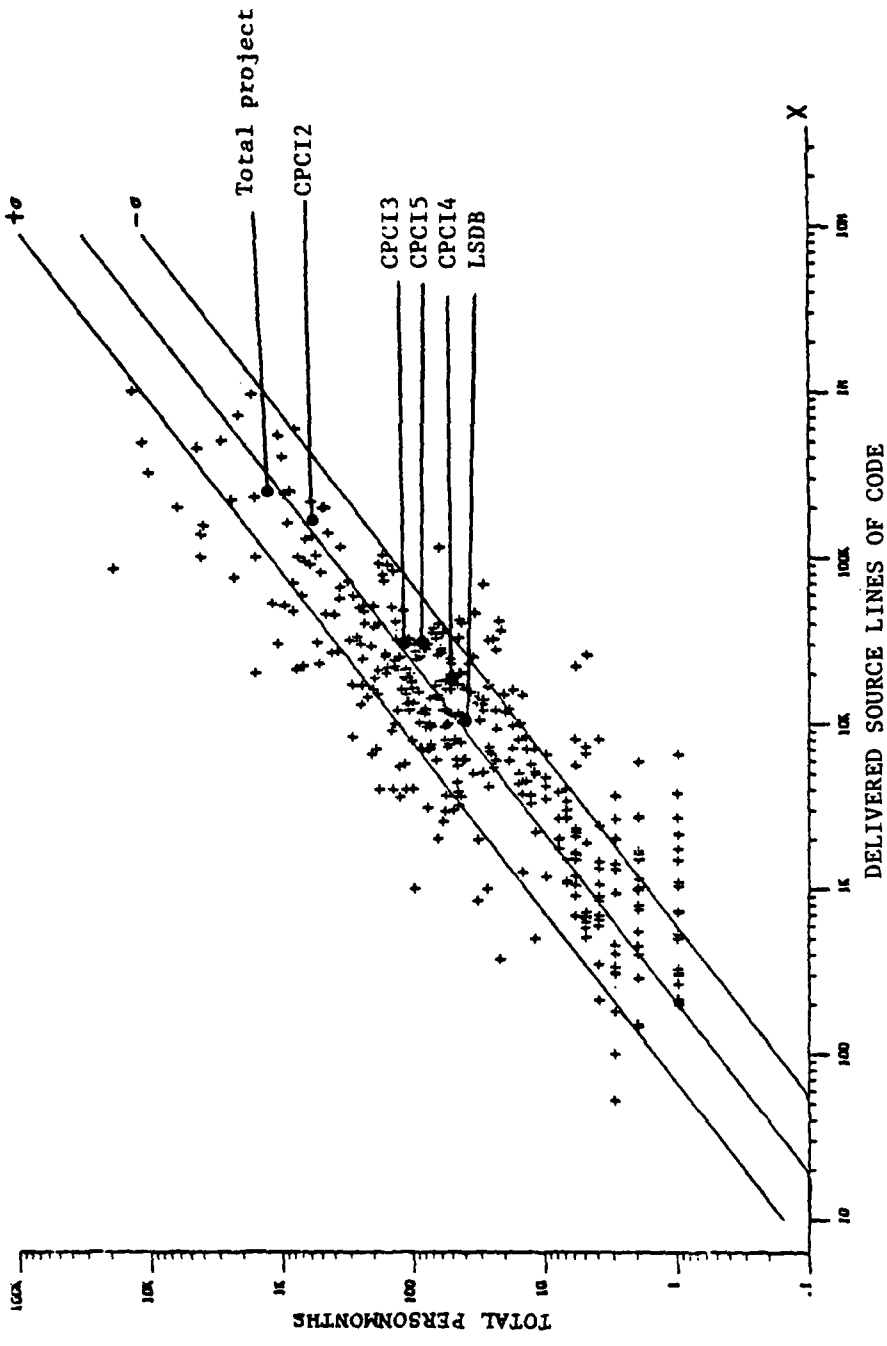


Figure 5. Comparison with other projects for total personmonths by lines of code

complete PAVE PAWS was typical of the time required to complete other projects of this size. That is, the datapoint for the total project fell next to the regression line, an indicator of the anticipated value for projects of similar size. Similar results were observed for each CPCI when plotted separately. The datapoint for the LSDB project (Curtis & Milliman, 1979), a modern progrmaming effort studied in this research project, also fell on the regression line when it was plotted into the figure. Thus, it does not appear from these data that modern programming practices lead to a reduced level of effort (personmonths) in developing software.

Datapoints for the PAVE PAWS and LSDB project were plotted into a scatterplot (Figure 6) similar to that in Figure 5 which contained only data from projects guided by modern progrmaming practices. Figure 6 indicates that the PAVE PAWS and LSDB projects were typical of the level of effort required in modern programming projects.

Data from the PAVE PAWS project were also compared against other projects in the RADC database on a productivity measure. This measure was developed by dividing the total delivered lines of code by the personmonths required to produce them. This is a gross measure of productivity and there are problems with its interpretation (Jones, 1978). Nevertheless, it is a measure which is often readily available for comparison among projects.

Personnel on the PAVE PAWS project produced an average of 186 lines of delivered source code per personmonth. For each CPCI the lines per personmonth were as follows: CPCI2-233, CPCI3-279, CPCI4-267, and CPCI5-300. The productivity of the total PAVE PAWS project is somewhat lower than that for each CPCI. This is probably due to the work categorized earlier as system level support software which involved development but was not delivered as part of CPCIs 2 through 5. Thus, these lines of code cannot contribute to the PAVE PAWS productivity figures. These productivity values are plotted into the RADC data in Figure 7, along with an indication of the productivity of the LSDB project. The datapoints for the PAVE PAWS project and its CPCIs fall near the regression line indicating that the level of productivity for this project was typical of that observed on software development projects of similar size. The same conclusion can be drawn for the LSDB project. The data for the PAVE PAWS and LSDB projects were also plotted into a graph containing productivity data for modern programming projects (Figure 8). These values fell close to the average for modern programming projects. Thus, average productivity seems to have been achieved on both the PAVE PAWS and LSDB

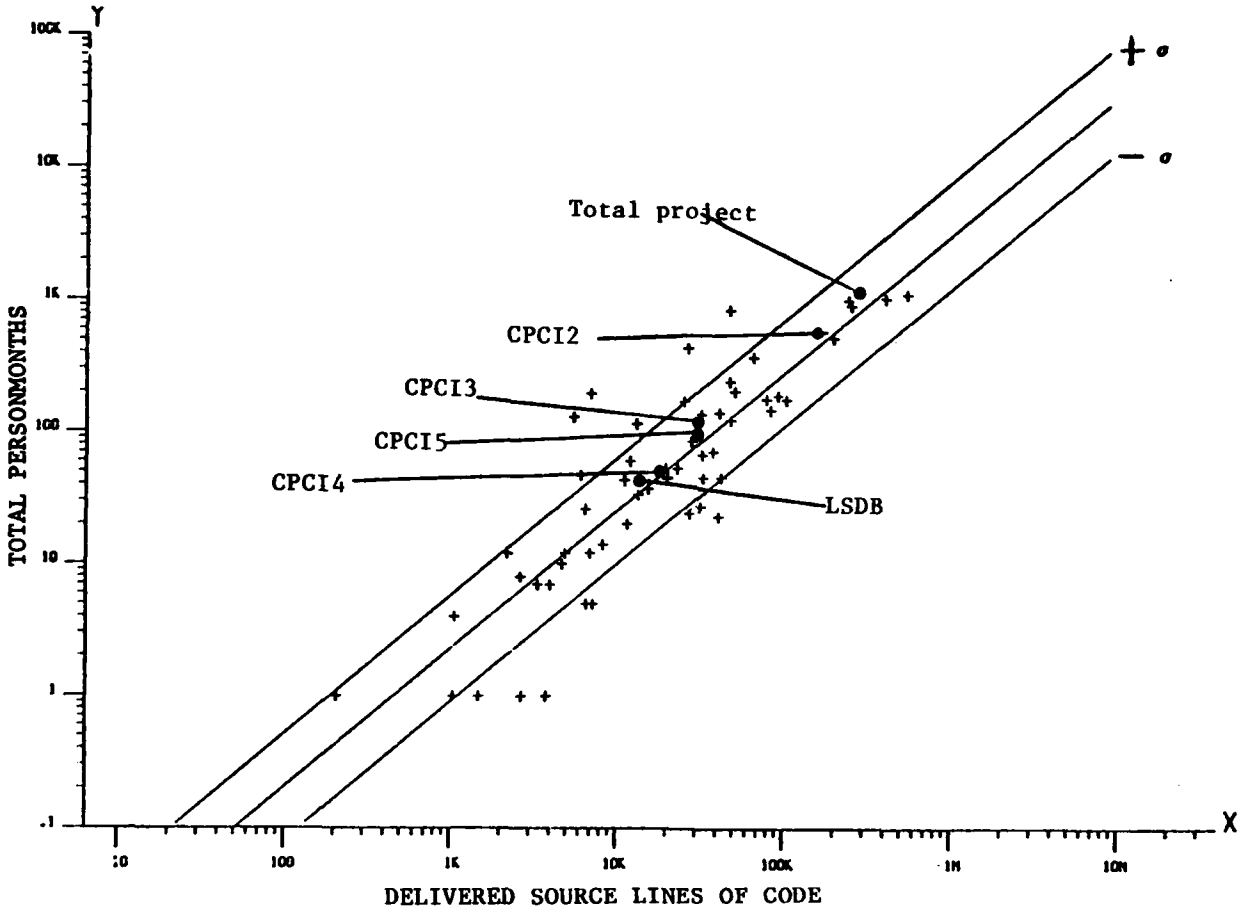


Figure 6. Comparison with modern programming projects for total personmonths

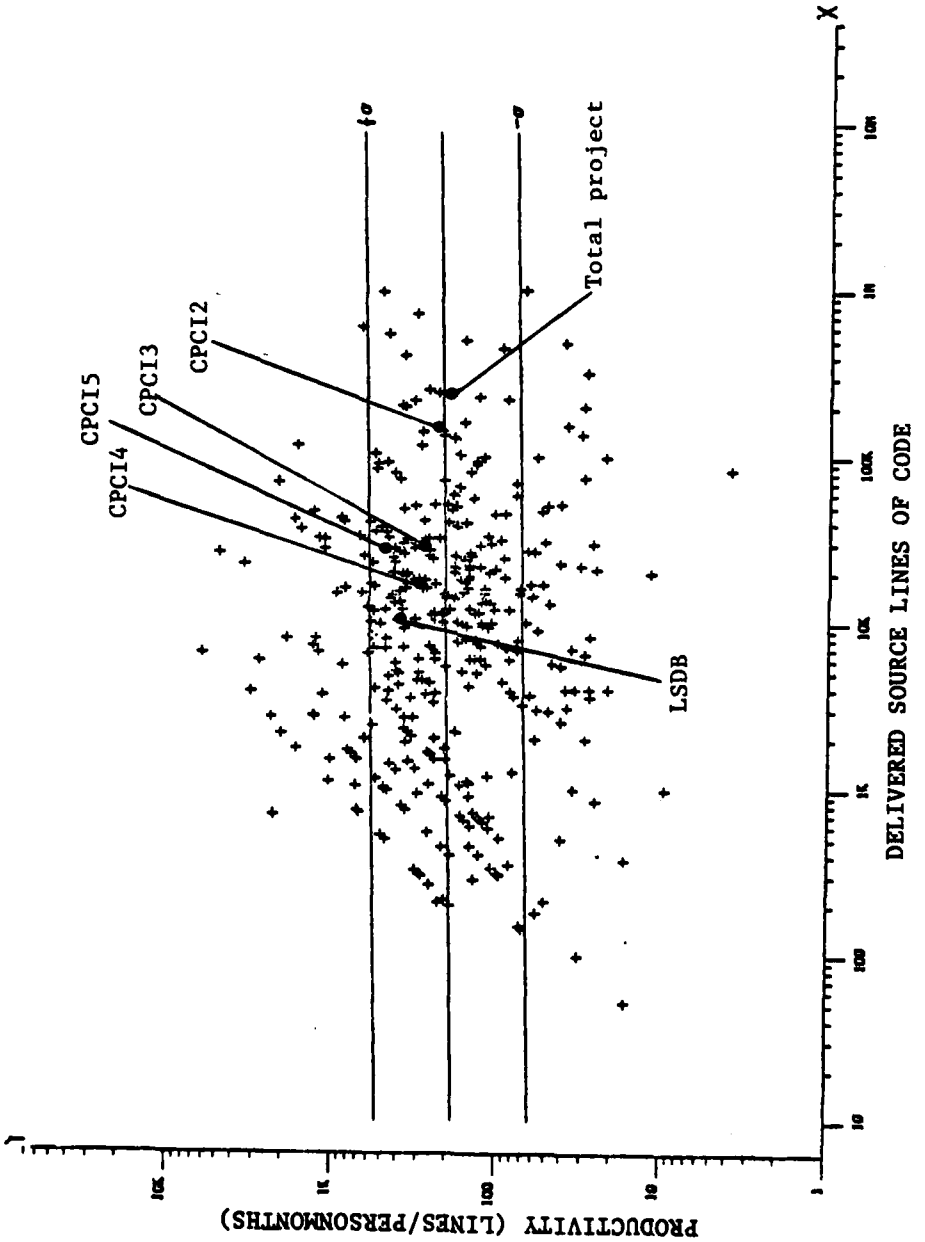


Figure 7. Comparison with other projects for productivity by lines of code

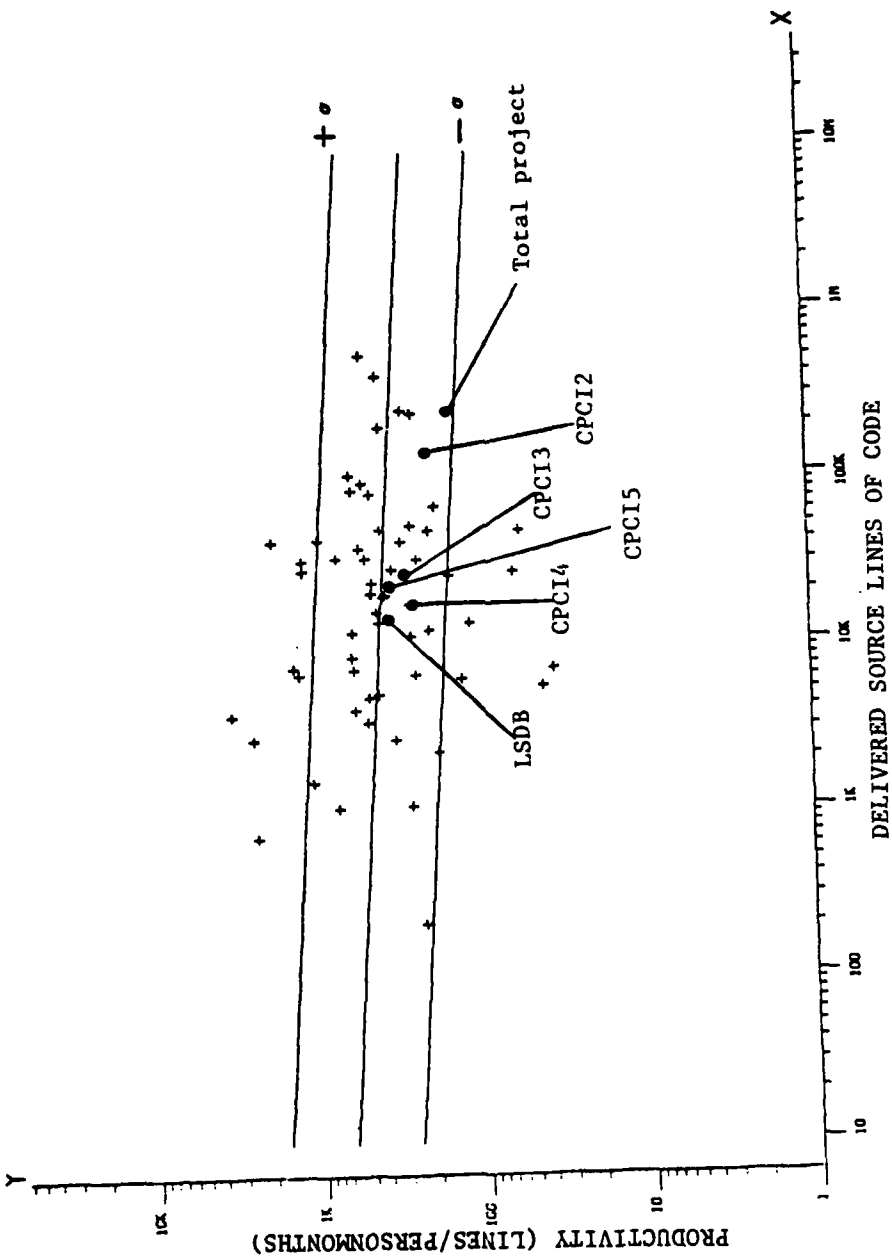


Figure 8. Comparison with modern programming projects on productivity

projects.

2.1.4 Compile summaries. From January through November 1978, 1756 compiles were performed through the Program Support Library (Table 6). The vast majority of these compiles were recorded after June 1978. However, as can be seen in the code progression chart for CPCIs 2 and 3 presented in Figure 9, the development effort had been largely completed by June and Final Qualification Tests had been performed. The same progression is true for CPCIs 4 and 5. The compiles captured by the PSL primarily involve cleaning up the code prior to delivery to the customer. The code durability chart for the total project presented in Figure 10 indicates that while 100% of the code had risen to the FIX level by June, some code had been drawn down to lower levels (e.g., PRG) for further work. It is unclear that the frequency of compiles by reason in Table 6 would hold true for earlier phases of the project. For instance, a larger relative frequency of INITIAL compiles would be anticipated during earlier phases. Of the compiles recorded, 69% were performed to correct algorithmic errors in the code (i.e., FUNCHG, COMPUTA, LOGIC, and SFNI), while only 2% involved INITIAL entries of code. Since most development runs were not recorded in the compiler summary file, there is little information which can be gleaned from this potentially valuable source of data for use in evaluating the effectiveness of modern programming practices on the PAVE PAWS project. However, the first four compile reasons correspond to frequent error types, and this relationship will be discussed in Section 2.2.4.

2.2 Error Analyses

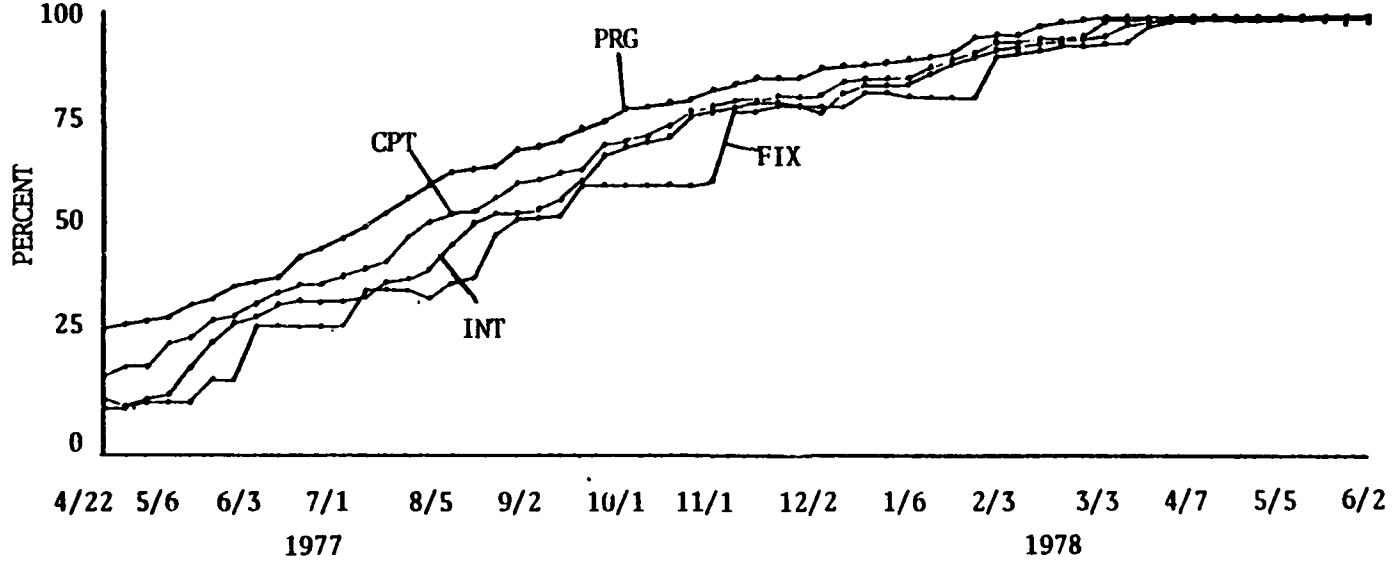
2.2.1 Frequency of Trouble Reports. The frequency of Trouble Reports by error category for each CPCI and the total project are presented in Table 7. There were 2099 Trouble Reports (TRs) filed for CPCIs 2 through 5 which had an interpretable error category. TRs which listed an error code corresponding to "unknown" or "reject" categories were not included in these analyses. CPCI2 accounted for 66% of the delivered code, and yet 86% of the TRs were reported against CPCI2. Thus, while the number of errors per thousand lines of code was only 2.93 for CPCI3, 5.25 for CPCI4, 4.59 for CPCI5, the rate for CPCI2 was 12.99, bringing the figure for the total project to 9.95.

The percent of errors falling in each category for each CPCI and the total project are presented in Table 8. The most frequent category was logic errors (43%), especially in CPCI2. Other frequently occurring errors were input/output

Table 6

Reasons for Compile by PSL Level

Compile Reason	PSL Level					Total
	PRG	CPT	INT	FIX	TST	
FUNCHG	358	48	81	16		503
COMPUTA	321	12	1			334
LOGIC	170	30	3	9		212
SFNI	159	6				165
OS	117					117
VERIFY	83		3		3	89
INTCHG	71	12				83
LISTING	46	4	4	15		69
DATA	57					57
KEY	31	3				34
SETUP	24			9		33
INITIAL	29			2		31
UNKNOWN	10					10
I/O	2			5		7
SINI	6					6
COMPILE	4					4
CPUOPT	1					1
PSL	1					1
Total	1490	115	92	56	3	1756


100
75
50
25
0
PERCENT
PRG
CPT
FIX
INT
4/22
5/6
6/3
7/1
8/5
9/2
10/1
11/1
12/2
1/6
2/3
3/3
4/7
5/5
6/2
1977
1978

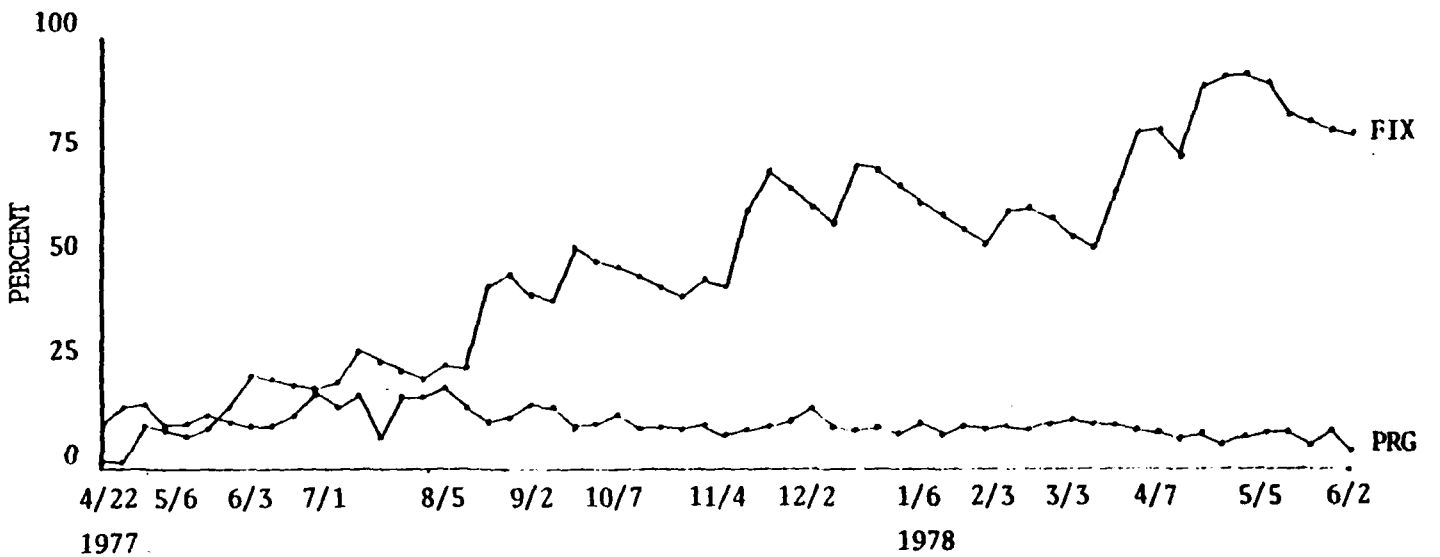


Figure 10. Durability chart for PAVE PAWS project

Table 7

Frequency of Trouble Reports by Error Category and CPCI

Error Category	CPCI				Total
	2	3	4	5	
Computation	64	5	0	2	71
Logic	826	23	31	31	911
I/O processing	228	5	2	43	278
Database	21	3	0	4	28
Unimplemented function	207	4	0	18	229
Unimplemented interface	97	6	1	4	108
Unspecified function required	27	2	19	0	48
Unspecified interface unsatisfied	26	2	0	0	28
Optimization needed	51	2	3	0	56
Redesign	182	28	27	19	256
Documentation change	38	3	0	)	41
Keypunch	9	0	1	0	10
Deck Setup	8	2	0	2	12
Configuration	22	0	0	1	23
Total	1806	85	84	124	2099

Table 8

Percentage of Trouble Reports by Error Category and CPCI

Error Category	CPCI				Total
	2	3	4	5	
Computation	4	6	0	2	3
Logic	46	27	37	25	44
I/O processing	13	6	2	35	13
Database	1	4	0	3	1
Unimplemented function	11	5	0	14	11
Unimplemented interface	5	7	1	3	5
Unspecified function required	1	2	23	0	2
Unspecified interface satisfied	1	2	0	0	1
Optimization needed	3	2	4	0	3
Redesign	10	33	32	15	12
Documentation change	2	4	0	0	2
Keypunch	1	0	1	0	1
Deck setup	1	2	0	2	1
Configuration	1	0	0	1	1

processing (13%), redesign (12%), and unimplemented function (11%). Table 9 presents the correlations between error profiles among the four CPCIs in order to determine their relatedness. Moderate correlations were observed among the CPCI profiles with an average interprofile correlation of 0.61. There was moderate consistency among the types of errors observed in developing the different CPCIs, but there were categories such as input/output processing and unimplemented functions in which the percentage of errors varied widely among CPCIs. In part, these differences may be related to differences in the nature of the functions being implemented in the CPCIs. For example, more I/O errors would be expected in CPCIs which must perform large amounts of input or output processing of data. CPCI5 primarily performed data reduction and had the largest percentage of I/O processing errors.

2.2.2 Comparison to other projects. In Figure 11 the PAVE PAWS error counts are plotted into the RADC database both for the total project and for each CPCI. The datapoints all fell near the regression line, suggesting that the PAVE PAWS project experienced a typical incidence of formally reported errors for projects of similar size. However, the point at which formal trouble reports begin to be generated may differ among projects. Nelson (1978) suggests that for most of the projects in this database such reporting does not begin until after unit testing is completed. Such a starting point was employed on PAVE PAWS and the project seems to have experienced an average number of errors for its size. When the number of errors in LSDB was plotted into this figure, it fell almost one standard error of estimate above the expected number of errors. However, it is difficult to accurately compare total errors across projects since they do not all begin formal trouble reporting procedures at the same point in project development.

2.2.3 Comparison of error categories. The error categories from the PAVE PAWS project could be compared against error categories from several recent RADC studies in which data were collected on error types. The projects covered in these studies include four projects from TRW (Thayer et al., 1976), and single projects from Raytheon (Williams et al., 1977), IBM (Baker, 1977), Boeing (Fries, 1977), Draper Labs (Rye et al., 1977), and Federal Electric (LSDB, Curtis & Milliman, 1979). Most of these studies employed an error classification scheme developed by Thayer et al. at TRW. In a few instances additional categories were added, and in two cases (TRW5 and LSDB) a reduced scheme was used which combined several categories (e.g., interface categories). The actual frequency of these errors by category is presented in Table 10.

Table 9

Correlation among Error Profiles by CPCI

CPCI	2	3	4	5
2	(.62)			
3	.69**	(.69)		
4	.65**	.83**	(.63)	
5	.52*	.54*	.41	(.49)

Note: Correlations in parentheses on the diagonal represent the average correlation for the CPCI's error profile with that of other CPCIs.

*$p \leq .05$

**$p \leq .01$

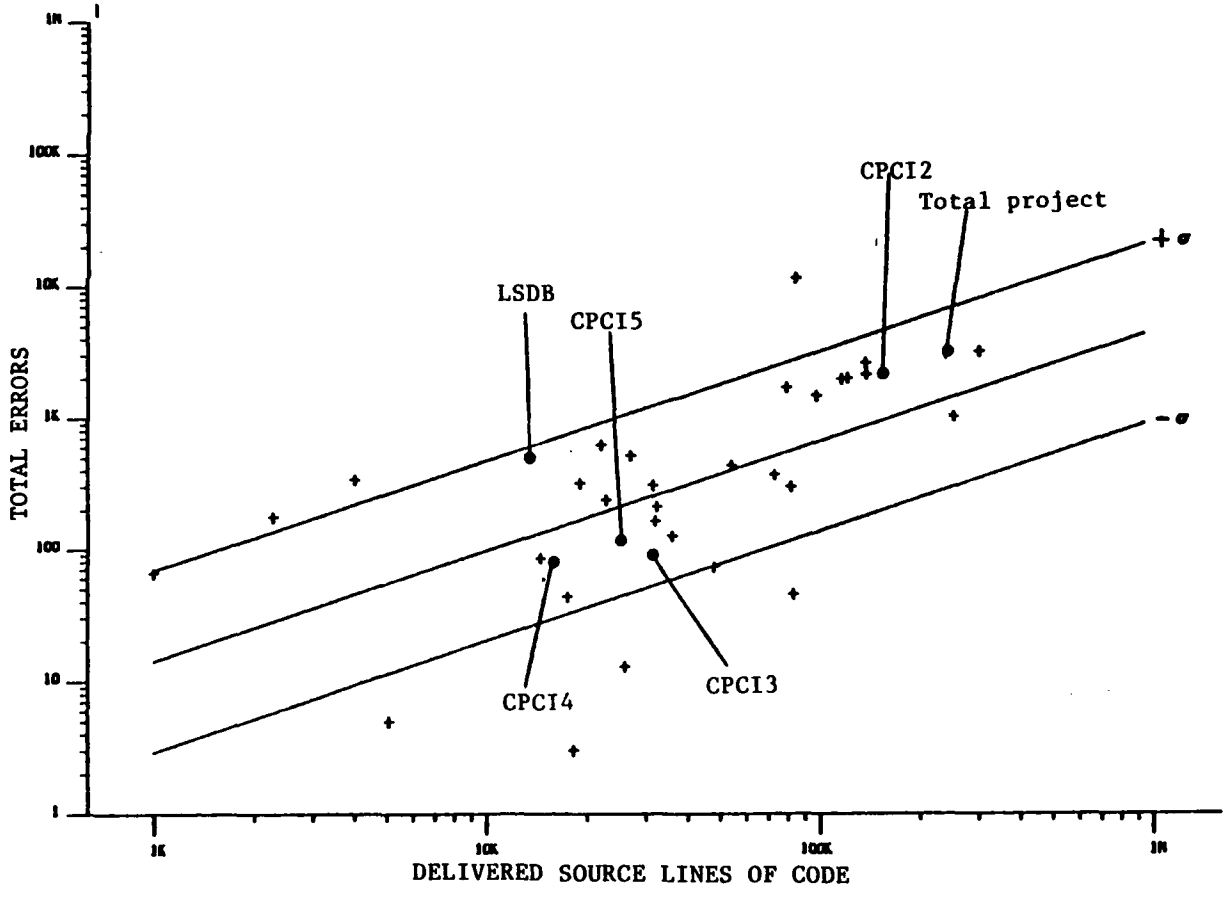


Figure 11. Comparison of total errors with other projects

Table 10

Frequency of Error Categories by Project

Error Category	PAVE PAWS	TRW 2	TRW 3	TRW 4	TRW 5	Raytheon	IBM	Boeing	Draper	LSDB
Computational	71	162	353	7	115	115	170	109	541	5
Logic	911	256	937	140	382	382	993	635	2217	98
Input/Output	278	156	719	36	21	21	454	28	287	29
Data handling		164	613	110	409	409	346	272	745	4
Operating system & support software		1	4	0	[37]	4	14	8	14	[156]
Configuration	[23]	16	83	0		18	19	12	1112	
Routine/routine interface		69	250	33		16	123	41	760	
Routine/system interface		5	28	0		17	38	3	683	
Tape processing interface	136	1	9	9	75	0	5	5	0	0
User interface		114	334	34		10	29	12	42	
Database interface		10	21	15		32	176	17	79	
User requested changes	312	0	0	111		764	188	161	780	
Preset database	[28]	111	370	9	[207]	162	310	67	355	[4]
Global variable/COOMPOOL definition		56	50	12		45	112	46	851	
Recurrent		24	80	0		39	820	148	280	
Documentation	41	172	196	23	15	15	796	27	727	0
Requirements compilance	277	10	47	0		10	32	144	57	0
Unidentified		80	178	0		77	236	30	66	88
Operator	22	86	118	0		15	26	158	0	124
Questions		5	49	0		3	102	19	0	
Requirements specification						11				
Hardware							246	32		
Requirements documentation							457			
In-house improvements									2123	
Total	2099	1498	4439	539	1261	2165	5692	1974	11719	508

Note: Blank spaces indicate the category was not included in the error breakdown for that project, while numbers in brackets indicate that a number of categories were combined into a single category for reporting on that project.

Unfortunately the PAVE PAWS error classification scheme was quite different from that used by other projects in the RADC database. The computation, logic, I/O, configuration, and documentation categories were roughly equivalent to the categories with similar names used in the other projects. However, in order to make comparisons possible it was necessary to reclassify several other categories, knowing that some inaccuracy would doubtless result from such redefinition. Some error categories from the other studies were also combined to facilitate comparisons. The five interface categories in other projects and the two interface categories in the PAVE PAWS project (Table 1, Codes 6 & 8) were collapsed into a common interface category. The preset database and global variable categories in other projects were combined and compared against the database category in PAVE PAWS. Unimplemented and unspecified functions on PAVE PAWS were compared to requirements compliance problems on other projects. Redesign and optimization categories on PAVE PAWS were compared to user requested changes on other projects although these categories may have included problems which should have been compared to requirements compliance errors. Keypunch and deck setup errors on PAVE PAWS were compared to operator errors on other projects. These changes underlie the reclassification of PAVE PAWS errors appearing in Table 10.

In order to compare error frequencies across projects, a subset of the error categories was selected for analysis. These categories are listed in Table 11. The user requested changes category was not included because not all projects seemed to include these among trouble reports while other categories were dropped which did not correspond to the reclassified PAVE PAWS error scheme. For purposes of comparison only 1787 of the 2099 PAVE PAWS trouble reports were studied. Because there were too many discrepancies in their categorization scheme, TRW5 and LSDB were not included in these reduced comparisons. Table 11 presents the frequences of these errors across projects while Table 12 presents the percent associated with their relative frequency of occurrence. There were some obvious areas of congruence such as the typically low percent of configuration errors and high percentage of logic errors. However, percentages of other categories varied in no clear pattern. Some of this variance may be due to differences in the way project personnel chose to classify certain types of problems.

To better compare error categories across projects, Table 13 presents correlations among the error profiles of the projects. The correlations observed were moderately high with an average correlation among error profiles of 0.70.

Table 11

Frequency of Error Categories Chosen for Comparison

Error Category	PAVE PAWS	TRW			Raytheon	IBM	Boeing	Draper
		2	3	4				
Computational	71	162	353	7	115	170	109	541
Logic	911	256	937	140	382	993	635	2217
Input/Output	278	156	719	36	21	454	28	287
Configuration	23	16	83	0	18	19	12	1122
Interface	136	199	642	91	75	371	78	1564
Database	28	167	420	21	207	422	113	1206
Documentation	41	172	196	23	15	796	27	727
Requirements compliance	227	10	47	0	10	32	144	57
Operator	22	86	118	0	15	26	158	0
Total	1787	1224	3515	318	858	3283	1304	7721

Table 12

Percent of Error Categories Chosen for Comparison

Error Category	PAVE PAWS	TRW			Raytheon	IBM	Boeing	Draper
		2	3	4				
Computational	4	13	10	2	13	5	8	7
Logic	51	21	27	44	45	30	49	29
Input/Output	16	13	21	11	2	14	2	4
Configuration	1	1	2	0	2	1	1	14
Interface	8	16	18	29	9	11	6	20
Database	2	14	12	7	24	13	9	16
Documentation	2	14	6	7	2	24	2	9
Requirements compliance	15	1	1	0	1	1	11	1
Operator	1	7	3	0	2	1	12	0

Table 13

Correlations among Error Profiles by Project

Project	PAVE PAWS	TRW 2	TRW 3	TRW 4	Raytheon	IBM	Boeing	Draper
PAVE PAWS	(.68)							
TRW 2	.47	(.67)						
TRW 3	.70*	.84**	(.72)					
TRW 4	.80**	.78**	.87***	(.78)				
Raytheon	.73*	.68*	.69*	.74*	(.73)			
IBM	.62*	.83**	.69*	.75**	.64*	(.67)		
Boeing	.90***	.49	.57	.73*	.86**	.55	(.67)	
Draper	.55	.63*	.65*	.82**	.78**	.63*	.58*	(.66)

Note: Correlations in parentheses on the diagonal represent the average correlation for the project's error profile with that of other projects.

*$p \leq .05$

**$p \leq .01$

***$p \leq .001$

TRW4 seemed to have the highest overall correlations with other projects. It appears that the profile of errors on the PAVE PAWS project is fairly typical of that experienced by other projects.

2.2.4 Error trends over time. The frequencies of errors in each category are presented by time period for CPCI2 in Table 14. This was the only CPCI with a sufficient number of errors to make trend comparisons meaningful. With the exception of the initial 4 months and the final 6 months of development on CPCI2, development time was divided into 3 month segments for the purpose of this analysis. Table 15 transforms each of these frequencies into the percentage of errors within each time period contained within each category. It is evident from these data that the percent of logical errors steadily increased over time. The percents of documentation and redesign errors decreased over time, with a brief flurry of design errors identified at the end of the project. Other categories such as I/O and configuration errors, appear to account for large percentages during the middle of the development period. It is clear from these data that the profile of error categories (in terms of relative frequencies) changes over time during software development.

The most frequent error categories during the last 6 months generally correspond to the most frequent compile listings reported during the same period (logic, redesign, unimplemented function). It is surprising that more computational errors were not reported given the frequency of compiles listing this reason in Table 6. However, a direct comparison is difficult given that Trouble Reports were supposedly generated only for errors captured during integration or higher levels of testing.

Table 14

Frequency of Error Categories by Time Period for CPCI2

	Months by year								
	1976	1977				1978			
Error Category	9-12	1-3	4-6	7-9	10-12	1-3	4-6	7-12	TOTAL
Computational		1	4	2	14	31	10	2	64
Logic	11	16	51	50	106	308	232	85	859
I/O processing	3	4	42	35	40	78	25	10	237
Database					3	13	5		21
Unimplemented function	6	3	8	17	22	96	53	11	216
Unimplemented interface	3	3	9	16	18	38	15		102
Unspecified function required					1	15	11	5	32
Unspecified interface satisfied		1	1		4	14	5	2	27
Optimization needed	5	1	3	13	22	12	3	1	60
Redesign	11	12	24	20	31	41	32	41	212
Documentation change	13	23	5		1	7			49
Keypunch						7	2		9
Deck setup			3	1	2		1	1	8
Configuration			7	8	5	3			23
Total	52	64	157	162	269	663	394	158	1919

Table 15

Percentages of Error Categories within Time Periods for CPCI2

	Months by Year							
	1976	1977				1978		
Error Category	9-12	1-3	4-6	7-9	10-12	1-3	4-6	7-12
Computational		2	3	1	5	5	3	1
Logic	21	25	32	31	39	46	59	54
I/O processing	6	6	27	22	15	12	6	6
Database					1	2	1	
Unimplemented function	11	4	5	10	8	15	13	7
Unimplemented interface	6	4	6	10	7	6	4	
Unspecified function required					0	2	3	3
Unspecified interface satisfied		2	1		2	2	1	1
Optimization needed	10	2	2	8	8	2	1	1
Redesign	21	19	15	12	12	6	8	26
Documentation change	25	36	3		0	1		
Keypunch						1	1	
Deck setup			2	1	1		0	1
Configuration			4	5	2	0		

3. CONCLUSIONS AND RECOMMENDATIONS

3.1 Conclusions

The PAVE PAWS software development project appears to have achieved average success on the outcomes studied in the data presented here. That is, it achieved the level of productivity and experienced the number of errors expected of projects of similar size. Although these data do not indicate increased productivity during software development through the use of modern programming practices, they do suggest that these practices contribute to the kind of control and management visibility which is required to guide software projects to successful completion on schedule and within budget. In particular, management found the reporting mechanism of the Program Support Library to be of tremendous value as a management information tool. Similarly, chief programmers believed that the chief programmer team structure contributed heavily to the overall performance and manageability of the PAVE PAWS project.

Unfortunately, the available data do not allow assessments of the separate contributions of each programming practice, or even of the cumulative effect versus the nonuse of such practices. However, project personnel prepared a description of how each practice was implemented on PAVE PAWS and assessed the success of each technique. This assessment appears in the Raytheon final report (1979) to RADC. The conclusions reached in that report were reiterated in our interviews with project personnel. Briefly, these conclusions were:

- Top-down design - makes the entire system design much more visible from early stages, contributes to logical progression in testing, and contributes to component independence.

- Structured coding - insures control flow will be much easier to comprehend, debug, and maintain, and does not appear to result in more code or poorly optimized code as is often claimed.

- Indented listings - aids programmer understanding and debugging.

- HIPO and PDL - HIPO charts seem to be valuable aids in system and subsystem design but are cumbersome when prepared for lower levels, while PDL has

numerous advantages at lower levels of system development. Constant updating may not prove cost-effective in a cost/benefit analysis; it should only be done periodically.

- Program support library - was an extremely valuable tool for providing configuration control, unit and integration testing, and management visibility.

- Chief programmer team - provided an organizing force to the work of project personnel. Should be staffed at a level of five or six people and the librarian may be shared with other teams.

- Structured walkthroughs - were beneficial when divided into two types. Design reviews were attended by project and customer personnel while code reviews were attended by rarely more than two others who could study the code in detail.

- Management information system - while PSL reports such as code progressions were of little assistance to programmers, they proved invaluable as a progress tracking tool for management.

From conclusions such as these it would appear that modern programming practices are not so much miraculous productivity aids as they are sound management practices. Thus, their use will reduce the risk and margin of error in predicting and controlling project outcomes.

3.2 Recommendations

3.2.1 Technical approach to life cycle research. As is evident in this report, the types of data necessary to evaluate the effectiveness of various programming techniques are not easy to collect. Although software development efforts generate an enormous assortment of numbers, research is not a scavenger hunt through the available data. Rather, data collection for research purposes should be designed from a theoretical model of the phenomena to be studied. It is important to identify the data to be collected from the factors in the model. Currently there are a number of critical areas in software development in need of theoretical models and evaluative data such as:

- Project sizing and costing

- Reliability prediction

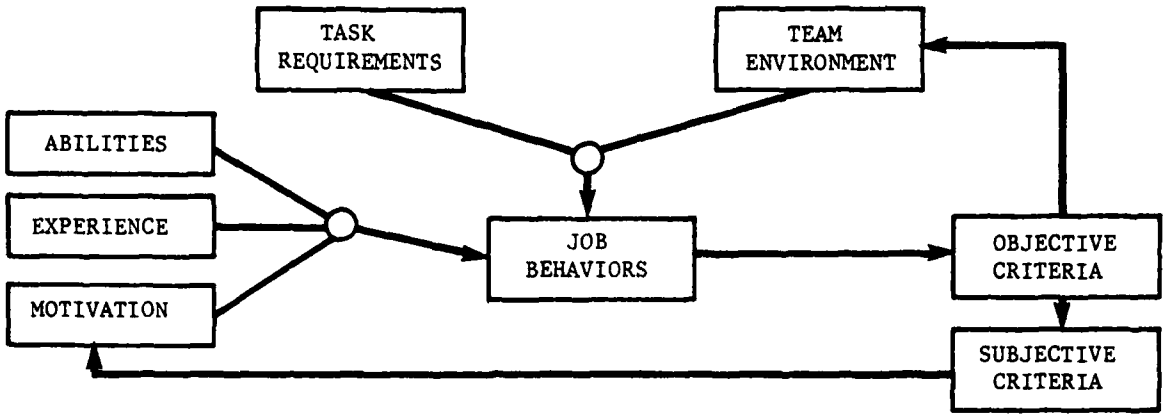
- Personnel selection and performance
- Programmer team organization
- Software design and testing strategies

In order to evaluate important questions in software engineering, a researcher must first determine the important factors affecting the criteria of interest and how they are related. For instance, Figure 12 presents a model of programmer performance and identifies some of the data which might be collected in order to evaluate various factors. Similarly, Figure 13 presents a model of team performance and some of the data which might be collected to evaluate this model.

There are two primary research strategies which can be employed in testing hypotheses from theoretical models depending on the type of controls which can be exercised over the variables affecting performance. In a laboratory situation, experimental controls can be exercised by manipulating the independent variable(s), holding all other situational factors constant, and minimizing the effect of individual differences among participants by randomly assigning them to different conditions of the experiment. The strongest causal statements can usually be made from such rigorously controlled experiments. On the other hand, research in field settings must usually rely on statistical controls to study the effects of different variables. Through the use of multivariate correlational methods such as structural equation models or time series analysis, underlying relationships can be teased from the data and different causal models can be compared to determine which is most consistent with the data. In using either experimental or statistical controls, it is always important to identify both the factors which may moderate the relationships observed and the populations to which the results can be generalized.

3.2.2 Development of a project database. In identifying data relevant to each factor in a theoretical model, there are a number of important considerations. First, the data should be collected at the appropriate level of explanation. The following are four possible levels of explanation:

- Programming environment
- Software development project
- Programming team



ABILITIES

- Abstract reasoning
- Verbal reasoning
- EDP knowledge

TASK REQUIREMENTS

- Programming practices
- Algorithm complexity
- Machine and language

TEAM ENVIRONMENT

- Leadership style
- Team organization
- Coworkers skills
- Organizational policies

EXPERIENCE

- Training
- Job history

OBJECTIVE CRITERIA

- Errors
- Time to completion
- Quality of code

MOTIVATION

- Interests
- Job attitudes

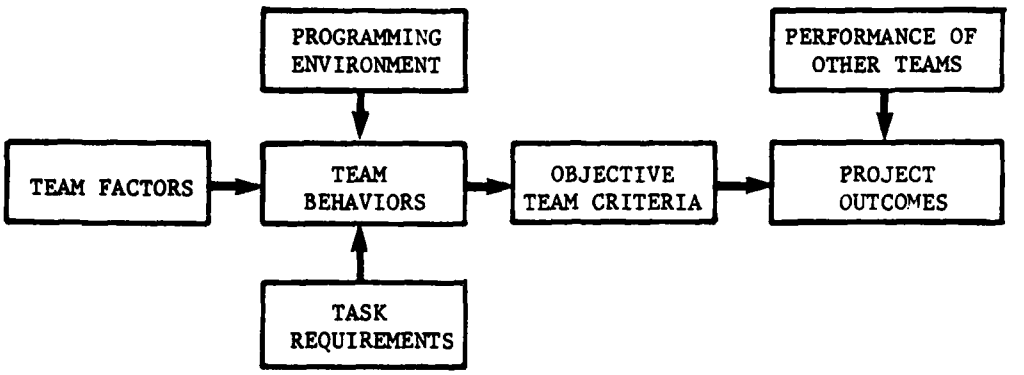
JOB BEHAVIORS

- Task accounting
- Absenteeism
- Runs
- Work habits

SUBJECTIVE CRITERIA

- Supervisor ratings
- Peer ratings

Figure 12. A model of individual programmer performance



PROGRAMMING ENVIRONMENT

- Organizational policies
- Access-turnaround time
- Tool availability

PERFORMANCE OF OTHER TEAMS

- Finished code
- Documentation
- Adherence to specs

OBJECTIVE TEAM CRITERIA

- Errors
- Schedule
- Costs
- Quality

TEAM FACTORS

- Leadership style
- Programming practices
- Skill mix
- Job attitudes

TEAM BEHAVIORS

- Runs
- Communication
- Integration

PROJECT OUTCOMES

- Budget
- Schedule
- Costs
- Software quality
- Software reliability

TASK REQUIREMENTS

- Algorithm complexity
- Language and machine
- Programming practices

Figure 13. A model of team productivity

- Individual programmers

Data collected at the project level are not sufficient by themselves to explain processes occurring at the level of the individual programmer. Thus, average lines of code per personmonth at the project level is not an adequate criteria for investigating the productivity of individual programmers. Performance at the project level involves effort spent integrating the work of programmers and programming teams above and beyond the work initially expended by programmers in developing their code. In analyzing data across levels of explanation it is important to specify rules for aggregation which identify how data at one level can be combined to explain processes occuring at the next higher level.

Performance itself is an ambiguously defined term. Rather than attempting to identify an ultimate criterion, a better approach would identify multiple criteria at several different levels of explanation. Identification of multiple criteria represents to software development managers the tradeoffs they must frequently make between schedule, budget, and quality (Figure 14). Regarding criteria relevant to schedule and budget, one can collect machine records (runs, errors, changes, cpu time, etc.), personnel and payroll records (manpower loadings, labor costs, absenteeism, etc.), and managerial performance ratings. In an RADC sponsored project, McCall, Richards, and Walters (1977) identified myriad attributes constituting software quality such as reliability, maintainability, portability, efficiency, etc.

The longitudinal collection of data and use of modeling techniques will allow an assessment of critical phases in the life of a development project and a programming team. A critical phase is a development step which influences important future outcomes. Management information regarding the outcomes of critical periods may serve as early warning flags concerning problems in specific areas.

Table 16 presents some of the domains of data which might be studied in a software research program. An important distinction is made between data which are objective versus those which are subjective. Objective data represent direct measurements of phenomena under consideration. Subjective data represent reports by project participants on their backgrounds, attitudes, perceptions of their work, etc., or in the case of managers, the performance of their subordinates. This objective-subjective distinction is especially important for interpreting criterion relationships, where subjective criteria represent more of a

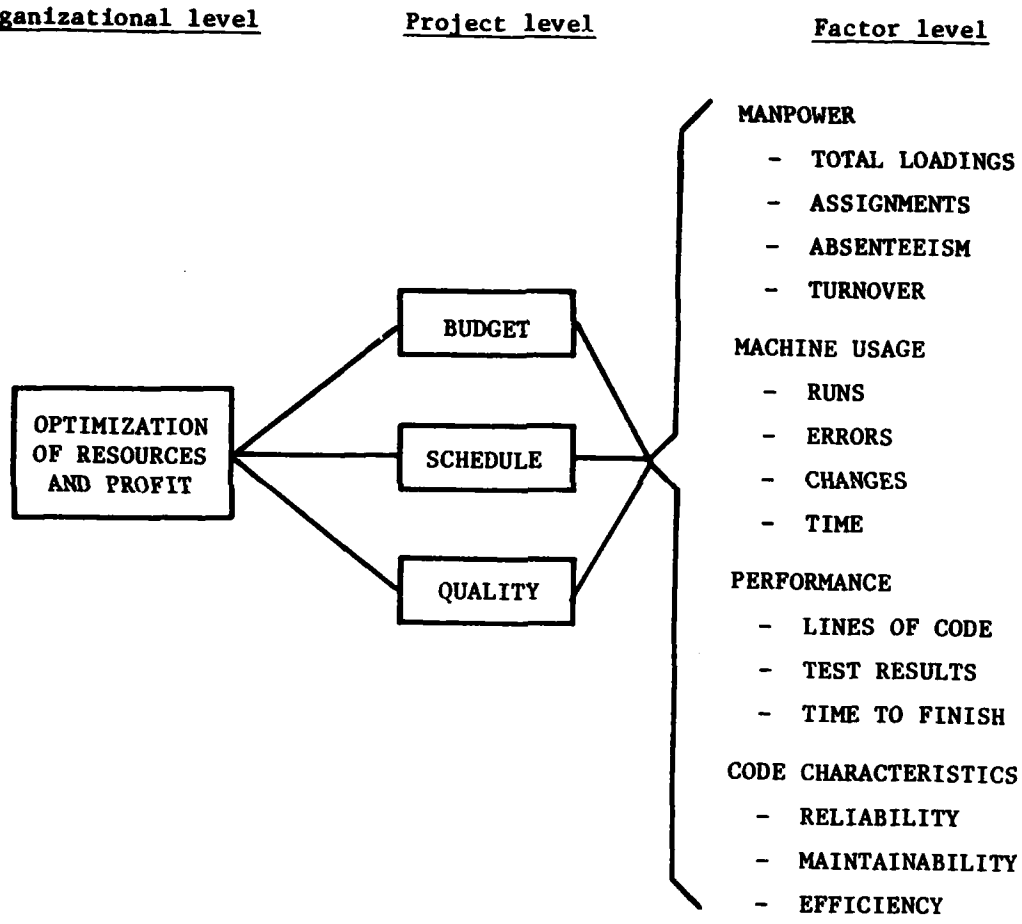


Figure 14. Multiple criteria model of project performance

Table 16

Domains of Data

Objective data	Subjective data
I. Team Factors	I. Individual Differences
A. Programming practices	A. Biographical
B. Size and composition	1. personal history
C. Task accounting	2. work experience
D. Physical environment	B. Intellectural
E. Communication patterns	1. abstract reasoning
	2. numerical ability
	C. Personality
	1. states-traits
	2. interests
	3. motivation
II. Task Complexity	II. Perceptions
A. Requirements	A. Management practices
B. System size	B. Work climate
C. Complexity of code	1. task
D. Module difficulty	2. leader
	3. work group
	4. organization
III. Criteria	III. Criteria
A. Machine records	A. Satisfaction
1. runs	B. Performance ratings
2. modifications	
3. errors	
4. CPU time	
B. Schedule completion	
C. Test results	
D. Personnel records	
1. absenteeism	
2. turnover	
E. Costs	
1. computer time	
2. manpower	

morale factor, and may themselves act as predictors of objective criteria in a reciprocal causation model. The primary domains of data to be collected are described in general terms below. Specific variables should be selected on the basis of a literature review.

There are a number of factors related to team performance (Figure 13) such as its size or personnel composition, the communication patterns established inside and outside of the group, programming practices such as the frequency and number of structured walkthroughs, and quality of the physical environment such as the proximity of team members. In addition, a task accounting system should be implemented which keeps periodic records of the time spent by team members on different aspects of their jobs. Post hoc measures of the complexity of modules should also be captured in the form of number of statements and complexity metrics (Halstead, 1977; McCabe, 1976) since these relate to task difficulty.

Three separate categories of individual differences variables might be collected (Figure 12). The first category involves biographical information describing personal history and work experience. The second category includes various tests of cognitive ability such as abstract reasoning and numerical ability which have been related to programmer performance. The final category of individual differences represents personality factors such as achievement motivation and the desire for routine versus complex work.

Variables which describe how actual conditions are perceived and interpreted by programmers offer insight into how these conditions affect individual and/or team performance. Such data has been described as the organizational or work climate. Data on how programmers perceive their work environment can be collected periodically on questionnaires. These data can be used to either describe the effects of certain programming management practices on team members, or to predict subsequent programmer or team performance.

Data should also be collected on code quality and complexity. Such metrics have been proposed by Halstead (1977), McCabe (1976), and McCall, Richards, and Walters (1977). These metrics can be assessed on the software at different stages of its development and used to predict outcomes at later stages (Figure 15). These metrics can also be used as measures of task difficulty in models of programmer or team performance.

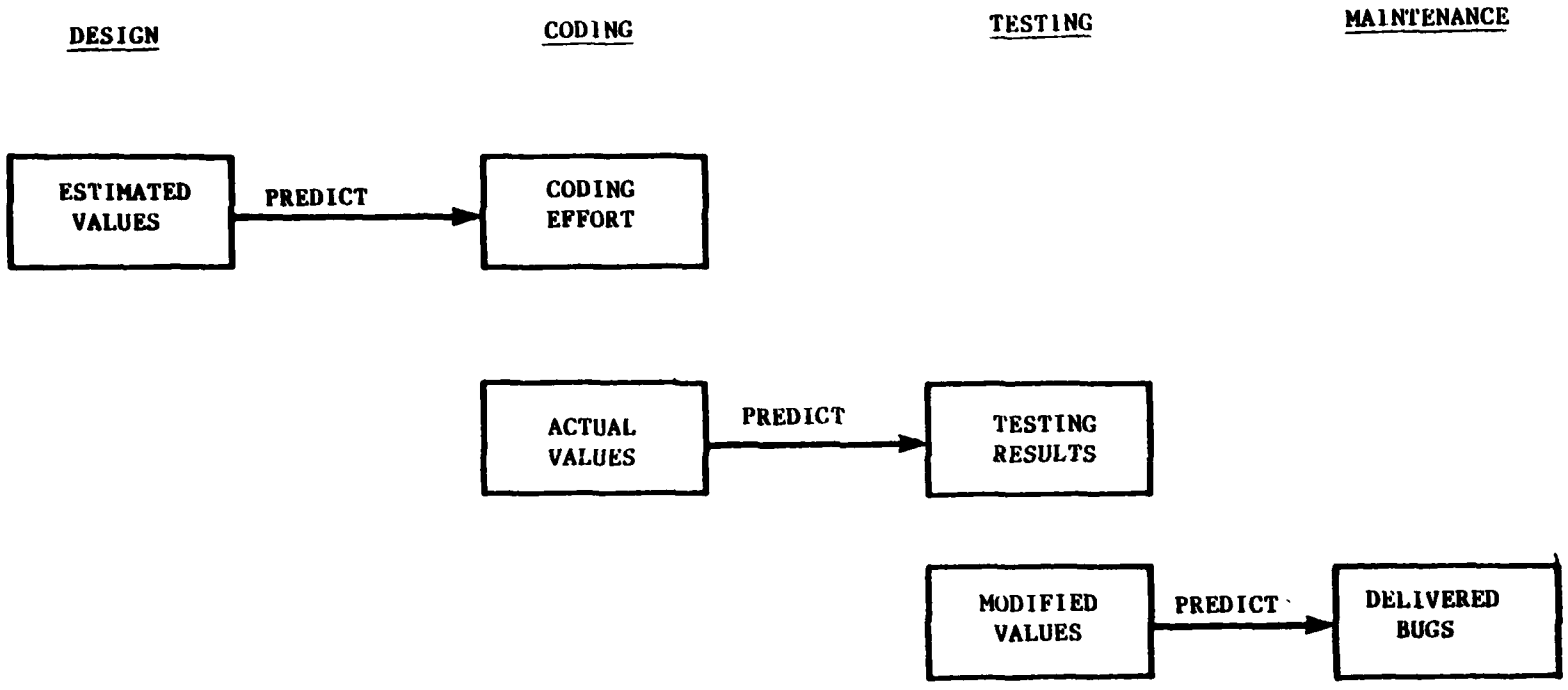


Figure 15. Predictive use of software metrics

Several criteria should be investigated at different levels of analysis as described in Figure 14. The two major classes of criteria to be collected are objective and subjective performance measures. Some objective performance measures can be collected on-line as each successive program is submitted to the computer. Thus, for each routine under development there will be a record for each run of the number and type of modifications made and the number and type of errors encountered. Over time a large database will emerge which can be analyzed for programming effectiveness at the level of either the individual programmer or the programming team. Other data will also be available in direct or derivable form regarding costs, completion schedules, test results, and personnel records.

The subjective criteria to be collected involve managerial performance evaluations of all programmers and of each programming team. Performance evaluations could be developed through critical incident techniques for both programmers and teams. Table 17 presents the anticipated frequency with which variables from different domains might be collected.

The following considerations are critical if useful evaluative data are to be obtained:

- Identify data relevant to each factor in a model
- Collect data at appropriate levels of explanation
- Identify multiple criteria
- Distinguish between objective and subjective data
- Distinguish between experimental and correlational data
- Identify appropriate time lags for data collection

Data collection on programming projects often interferes with the programming task or meets opposition from team members. This problem can be counteracted by developing measurement tools embedded in the system which are invisible to programmers. Such tools would produce more reliable data since they cannot be forgotten, ignored, or incorrectly completed as forms frequently are. A program support library can report module size, number of runs, and other summary information at regular intervals, and check project status to alert the manager of milestones. When these collection mechanisms are imposed unobtrusively on programming projects, their use is more likely to gain support, and better data

Table 17

Frequency of Data Collection

Variable	Frequency of Collection
Team Factors	monthly
Task Complexity: initial rating post hoc measures	during startup as module completed
Machine Records	daily
Schedule completion	as available
Testing Results	as available
Personnel Records	weekly
Costs	monthly
Individual Differences	during startup
Management Practices Survey	bi-monthly
Work Climate	every 6 months
Satisfaction	every 6 months
Performance Ratings	every 6 months

will be available both for management and research.

As the data base increases, data should be periodically edited to insure accuracy. At the same time the development of composite scores can begin where appropriate; for instance, the development of a reduced set of composites from the individual differences data. While such composites should be established on conceptual grounds, multivariate techniques such as factor analysis are available to aid the process of data reduction. That is, only conceptually distinct sets of data (e.g., biographical items or job perceptions) should be entered into an empirical data reduction technique. For data which have been collected longitudinally, the reliability of scores to be used for further analysis can be tested for stability as well as internal consistency.

In summary the following guidelines for software life cycle research are proposed:

- Begin with a theoretical model
- Identify an appropriate research strategy
- Appoint someone responsible for data collection
- Collect data which is
  - appropriate to the level of explanation
  - objective
  - longitudinal
- Identify multiple criteria
- Hire a good statistician

The major contribution of such a research program would be the wealth of information it would generate concerning the management of large software development efforts, particularly of the kind sponsored by DOD and other federal agencies. Some of the specific outcomes from such projects would be:

- Guidelines for developing and interpreting Management Information Systems (MIS) for tracking progress in large software development efforts.
- Guidelines for organizing and starting up programming projects.
- Predicting software quality and reliability.

- Guidelines for selection and placement of programmers.

- Additional refinement of software life-cycle for sizing and costing software development e

The data collected and analyzed on the PAVE PAWS LSDB projects have provided initial examples of how s research might be approached. The results have been encouraging, and more comprehensive databases are nee future progress.

4. REFERENCES

Baker, F.T. Chief programmer team management of production programming. IBM Systems Journal 1972, 11, 56-73.

Baker, F.T., & Mills, H.D. Chief programmer teams. Datamation, 1973, 19 (12), 58-61.

Baker, W.F. Software data collection and analysis: A real-time system project history (Tech. Rep. RADC-TR-77-192). Griffiss AFB, NY: Rome Air Development Center, 1977. (A041644)

Barry, B.S., & Naughton, J.J. Structured Programming Series (Vol. 10) Chief Programmer Team Operations Description (RADC-TR-7400-300-Vol.X). Griffiss AFB, NY: Rome Air Development Center, 1975 (NTIS No. AD-A008 861).

Belford, P.C., Donahoo, J.D., & Heard, W.J. An evaluation of the effectiveness of software engineering techniques. In Proceedings of COMPCON '77. New York: IEEE, 1977.

Black, R.K.E. Effects of modern programming practices on software development costs. In Proceedings of COMPCON '77. New York: IEEE, 1977.

Boehm, B.W. Software and its impact: A quantitative assessment. Datamation, 1973, 19 (5), 48-59.

Brown, J.R. Modern programming practices in large scale software development. In Proceedings of COMPCON '77. New York: IEEE, 1977.

Curtis, B. & Milliman, P. A matched project evaluation of modern programming practices (2 vols.). Griffiss AFB. NY: Rome Air Development Center, 1980. RADC-TR-80-6

DeRoze, B.C. Software research and development technology in the Department of Defense. Paper presented at the AIIE Conference on Software, Washington, D.C., December 1977.

DeRoze, B.C., & Nyman, T.H. The software life cycle - A management and technological challenge in the Department of Defense. IEEE Transactions on Software Engineering, 1978, 4, 309-318.

Dijkstra, E.W. Notes on structured programming. In O.J. Dahl, E.W. Dijkstra, & C.A.R. Hoare (Eds.), Structured Programming. New York; Academic Press, 1972.

Duvall, L.M. The design of a software analysis center. In Proceedings of COMPSAC '77. New York: IEEE, 1977.

Fries, M.J. Software error data acquisition. (Tech. Rep. RADC-TR-77-130). Griffiss AFB, NY: Rome Air Development Center, 1977. (A039916)

Halstead, M.H. Elements of Software Science. New York; Elsevier North-Holland, 1977.

Jones, T.C. Measuring programming quality and productivity. IBM Systems Journal, 1978, 17 (1), 39-63.

Katzen, H. Systems Design and Documentation: An Introduction to the HIPO Method. New York: Van Nostrand Reinhold, 1976.

Lyons, E.A., & Hall, R.R. ASTROS: Advanced Systematic Techniques for Reliable Operational Software. Vandenburg AFB, CA: Space and Missile Test Center, 1976.

McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering, 1976, 2, 308-320.

McCall, J.A., Richards, P.K., & Walters, G.F. Factors in software quality (Tech. Rep. 77C1S02). Sunnyvale, CA: General Electric, Command and Information Systems, 1977.

Mills, H.D. Mathematical foundations for structured programming. In V.R. Basili and T. Baker (Eds.), Structured Programming. New York: IEEE, 1975.

Myers, G.J. Composite/Structured Design. New York: Van Nostrand Reinhold, 1978.

Myers, W. The need for software engineering. Computer, 1978, 11 (2), 12-26.

Nelson, R. Software data collection and analysis. NY: Rome Air Development Center, 1978.

Parnas, D. On the criteria for decomposing systems into modules. Communications of the ACM. 1972, 15, 1053-1058.

Putnam, L.H. A general empirical solution to the macro software sizing and estimating problem. IEEE Transactions on Software Engineering, 1978, 4, 345-361.

Raytheon. PAVE PAWS Modern Programming Data Collection System: Final Report. Griffiss AFB; NY: Rome Air Development Center, 1979.

Rye, P., Ostanek, W., Bamburger, F., Broden, N., & Goode, J. Software systems development: A CSDL case history (Tech. Rep. RADC-TR-77-213). Griffiss AFB, NY: Rome Air Development Center, 1977. (A042186)

Salazar, J.A., & Hall, R.R. ASTROS Advanced Systematic Techniques for Reliable Operational Software: Another Look. Lompac, CA: Vandenburg, AFB, Space and Missile Test Center, 1977.

Stay, J.F. HIPO and integrated program design. IBM Systems Journal, 1976, 15, 143-154.

Stevens, W.P., Myers, G.J., & Constantine, L.L. Structured design. IBM Systems Journal, 1974, 13, 115-139.

Tausworthe, R.C. Standardized Development of Computer Software (2 vols.). Englewood Cliffs, NJ: Prentice-Hall, 1979.

Thayer, T.A., et al. Software reliability study (RADC-TR-76-238). Griffiss AFB, NY: Rome Air Development Center, 1976. (A030798)

Walston, C.E., & Felix, C.P. A method of programming measurement and estimation. IBM Systems Journal, 1977, 18 (1), 54-73.

Willman, H.E., Beaureguard, A.A., James, T.A., & Hilcoff, P. Software systems reliability: A Raytheon project history (Tech. Rep. RADC-TR-77-188). Griffiss AFB, NY: Rome Air Development Center, 1977. (A040992)

Yourdon, E., & Constantine, L.L. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Englewood Cliffs, NJ: Prentice-Hall, 1979.

Yourdon Report (Vol. 1, No. 9). New York: Yourdon, Inc. 1976.

ACKNOWLEDGEMENTS

We sincerely appreciate the help we received in implementing this research from Don Roberts, our contract monitor, and Nancy Hall, our liason at IBM. We also appreciate the help of our colleagues Phil Milliman in data analysis and Beverly Day for manuscript preparation. The support and encouragement of Lou Oliver has also been important in implementing this research.

Appendix A

Project Summary Forms

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

GENERAL CONTRACT/PROJECT SUMMARY

1. Type of Contract: FFP ____ CPFF ____ OTHER FPIF

2. Total Cost (Actual or Estimated) $5.0M (CPCI's effort only)

3. Level of Subcontracting None

4. Project Environment

Dev. Team Collocated with User?	No
Dev. Team Collocated with Computer?	Yes
Dev. System Same as Operational System?	Yes
Test & Integration Separate Organization?	Yes

5. Project Description

Engineering support plus software design, fabrication, and test for

(1) PAVE PAWS Tactical Software (CPCI 2) which is a real-time system including input and output interfaces with the PAVE PAWS Radar Controller (RCL-CPCI 6) via the PAVE PAWS Operating System (PPOS-CPCI 1). The system has strict storage and throughput goals.

(2) PAVE PAWS Simulation Software (CPCI 3) which is a real-time system with the same interfacing requirements as above.

(3) PAVE PAWS Tactical Scenario Generator (CPCI 3) which is a non-real-time data base maintenance tool used to prepare scenario files used to drive Simulation.

(4) PAVE PAWS Data Reduction (CPCI 5) which is a non-real-time reduction system for a large variety of recording which is done by both CPCI 2 and CPCI 3.

(5) PAVE PAWS Program Support Library (PSL-CPCI 4) which provides the basic software library services in a topdown structured environment.

6. Project Start Date 04/12/76 Est. End Date 04/12/78

7. Estimated Number of Project Personnel

Management	5	Systems Engineering	6
Chief Programmer	6	Functional Test	10
Support	6	Dev. Programming	31

8. Estimated Number of CPC's 48

9. Estimated Number of Pages of Documentation

Requirements (Part I)	1460	Test Reports	1200
Specifications (Part II)	3400	User Manuals	900
Test Specifications	2000	Other	600

10. Estimated Total Number of Instructions N/A Cards 135K

11. Estimated Number of Different Input Formats N/A

12. Estimated Number of Different Output Formats N/A

13. Estimated Total Man/Months

Management	85	Programming	630
Support	102	Test	170
Engineering	102		

14. Estimated Total Computer Time (HRS) 7000 Hours
(wall clock on dedicated computer)

Contact B. Scheff (Raytheon)

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

MANAGEMENT METHODOLOGY SUMMARY

1. **Management Procedures/Tools Used**

   PAVE PAWS Program Support Library (PSL) reporting
   PAVE PAWS Trouble Report Procedures
   Program Control Management System (PCMS - Financial)

2. **Documentation Available at CDR:**

   a. Development Specification (Part I) - CPCI 2
   b. Development Specification (Part I) - CPCI 3
   c. Development Specification (Part I) - CPCI 4
   d. Development Specification (Part I) - CPCI 5
   e. Product Specification (Part II) - CPCI 2
   f. Product Specification (Part II) - CPCI 3
   g. Product Specification (Part II) - CPCI 4
   h. Product Specification (Part II) - CPCI 5

   NOTE: All above documents provided to customer.

3. **Formal Reviews and Schedule**

	Date	
a. CPCI 2	PDR 8/76	CDR 1/77
b. CPCI 3	PDR 8/76	CDR 1/77
c. CPCI 4	PDR 7/76	CDR 9/77
d. CPCI 5	PDR 8/76	CDR 1/77

4. **AF Regulations, Manuals, and Military Standards Under Which** Development Will Be Conducted.

   MIL-STD-483
   MIL-STD-490
   MIL-STD-1521

5. Description of Deliverable Software

Refer to GENERAL CONTRACT/PROJECT SUMMARY, Item 5, for an overview of the technical content of deliverable software. All software will be delivered in a PSL form (either disk or checkpoint tape).

6. Reference Measurement Gathering Procedures

Clarification required.

Contact B. Scheff (Raytheon)

SYSTEM PAVE PAWS (Data Collected Against) DATE 10/07/77

DESIGN AND PROCESSOR SUMMARY

1. Target Computer(s) CDC CYBER 174-12
(same as development computer)

2. Processing Environment

1 Card Reader (CDC 405)
2 Line Printers (CDC 580-12)
3 Disk Drives (CDC 844-21)
6 CRT's (CDC 774-1)
1 Plotter (Gould)
6 Tape Drives (CDC 669-2)

3. Configuration: Hands on X Batch X Remote ____ On-line _

4. Operating System(s) Version Nos. 1.0 as modified (PPOS)

5. Compiler Version(s) JOVIAL J3

6. Assembler(s) COMPASS

7. Est. Percent: JOVIAL 85 COMPASS 15

8. Automated Software Tools Used: PAVE PAWS PSL

9. Design Standards

- MIL-STD-483, Appendix VI
- IBM FSD Software Standards (33-09)

10. Programming Standards

- PAVE PAWS Green Sheets
- PAVE PAWS Computer Development Plan

11. Programming Techniques Employed:

Topdown Design	X	HIPO	X
Chief Programmer	X	Structured Code	X
Librarian	X	Structured Walk Thru	X
Topdown Test	X	Other - PDL	X

A002(d)
R&D-111-RADC

12. List Existing Programs/CPC's to be Used Standard commercial software

13. Estimated Turnaround Time (HRS): Batch 2 Hours

Contact B. Scheff (Raytheon)

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.